来自 http://simple-is-better.com/news/361

博客刚开，打算做一个 Pygame 的系列，翻译自 Will McGugan 的 Beginning Game Development with Python and Pygame –From Novice to Professional》，有兴趣的朋友可以搜一下，有英文版的 PDF 可以下载。其实也不是翻译，把精华摘出来，共同学习。

看这个系列需要有 Python 的基础知识，虽然一开始想写一篇 Python 概要的，实在是很庞杂，而且有那么多好的 Python 基础教程，自己就不多插一脚了吧。入门的话，有 Python 入门，详尽的话，可以看看 Python 核心编程或者 Python 编程金典，然后 IBM 上的“可爱的 Python”系列也很不错，可以扩展一下思维。

OK，让我们开始吧~

Pygame 的历史

Pygame 是一个利用 SDL 库的写就的游戏库，SDL 呢，全名 Simple DirectMedia Layer，是一位叫做 Sam Lantinga 的大牛写的，据说他为了让 Loki（致力于向 Linux 上移植 Windows 的游戏的一家大好人公司，可惜已经倒闭，唉好人不长命啊……）更有效的工作，创造了这个东东。

SDL 是用 C 写的，不过它也可以使用 C++进行开发，当然还有很多其它的语言，Pygame 就是 Python 中使用它的一个库。Pygame 已经存在很多时间了，许多优秀的程序员加入其中，把 Pygame 做得

越来越好。

安装 Pygame

你可以从 www.pygame.org 下载 Pygame，选择合适你的操作系统和合适的版本，然后安装就可以了（什么，你连 Python 都没有？您可能是不适合看这个系列了，不过如果执意要学，很好！快去 www.python.org 下载吧 ）。 一旦你安装好，你可以用下面的方法确认下有没有安装成功：

```
>>>importpygame
>>>printpygame.ver
1.9.1release
```

你的版本可能和我不同，这没关系。我所翻译的这本书上的版本还是1.7.1的……所以如果有些过时的不合时宜的东西，千万不要客气请指出来！

若说为什么要介绍这么一个“过时”的东西，真正的知识是不会过时的，只有技术才会。这里主要是依靠 Pygame 来介绍的游戏开发的方方面面，并不是说咱就可以靠这个做出什么伟大的游戏了（当然也不是说不可以 ）！

另外说一下，就产品而言，Pygame 更致力于2D 游戏的开发，也就是说，你可以用 Pygame 写一个植物大战僵尸，但是写一个魔兽世界则相当困难……请不要做出鄙夷的目光，底层的东西永远是相通的，而且对于新手而言，从简单的2D 入手才是正途。

使用 Pygame

Pygame 有很多的模块，下面是一张一览表：

模块名	功能
pygame.cdrom	访问光驱
pygame.cursors	加载光标
pygame.display	访问显示设备
pygame.draw	绘制形状、线和点
pygame.event	管理事件

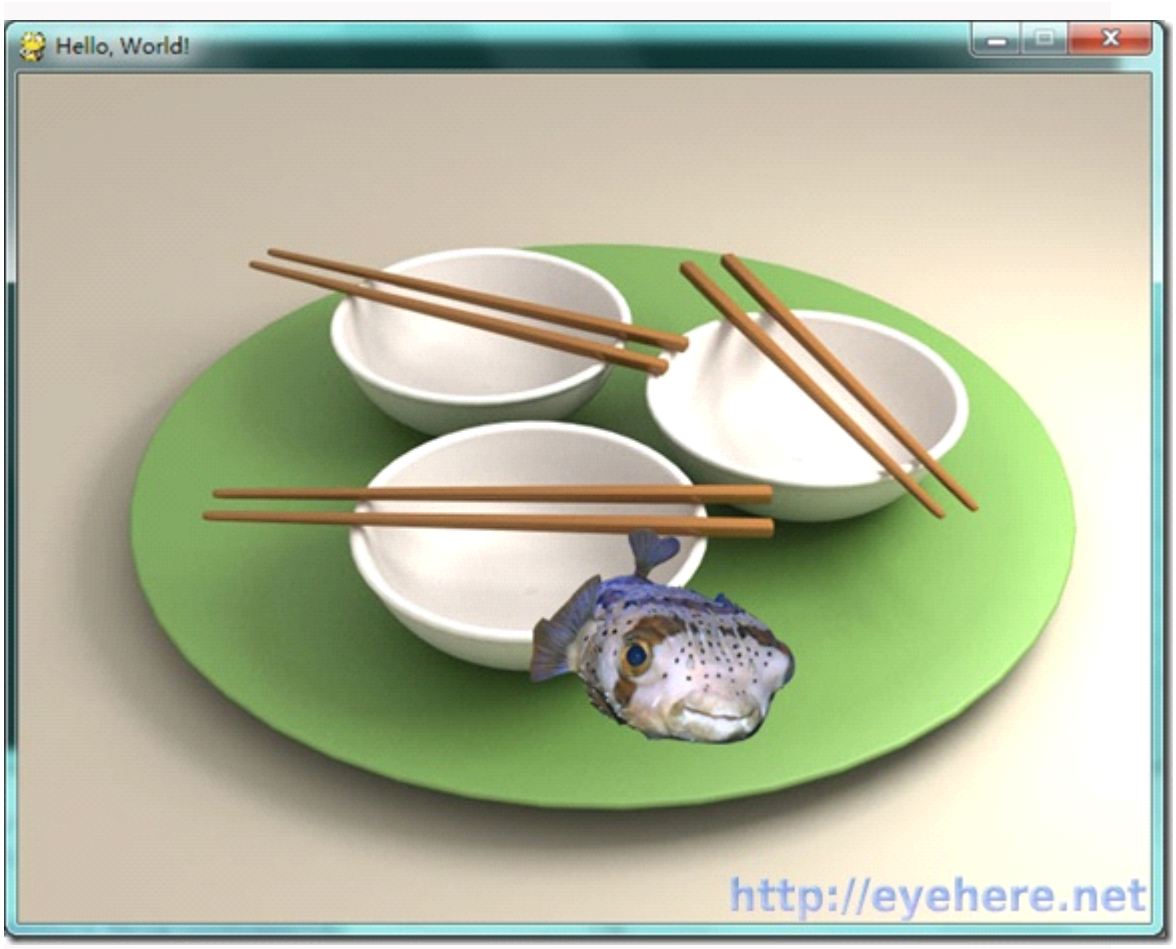
pygame.font	使用字体
pygame.image	加载和存储图片
pygame.joystick	使用游戏手柄或者 类似的东西
pygame.key	读取键盘按键
pygame.mixer	声音
pygame.mouse	鼠标
pygame.movie	播放视频
pygame.music	播放音频
pygame.overlay	访问高级视频叠加
pygame	就是我们在学的这个东西了……
pygame.rect	管理矩形区域
pygame.sndarray	操作声音数据
pygame.sprite	操作移动图像
pygame.surface	管理图像和屏幕
pygame.surfarray	管理点阵图像数据
pygame.time	管理时间和帧信息
pygame.transform	缩放和移动图像

有些模块可能在某些平台上不存在，你可以用 None 来测试一下。

```
ifpygame.fontisNone:
print"The font module is not available!"
exit()
```

新的 Hello World

学程序一开始我们总会写一个 Hello world 程序，但那只是在屏幕上写了两个字，现在我们来点更帅的！写好以后会是这样的效果：




```
#!/usr/bin/env python
background_image_filename='sushiplate.jpg'
mouse_image_filename='fugu.png'
#指定图像文件名称
importpygame
#导入 pygame 库
frompygame.localsimport*
#导入一些常用的函数和常量
fromsysimportexit
#向 sys 模块借一个 exit 函数用来退出程序
pygame.init()
#初始化 pygame,为使用硬件做准备
screen=pygame.display.set_mode((640,480),0,32)
#创建了一个窗口
pygame.display.set_caption("Hello, World!")
#设置窗口标题
background=pygame.image.load(background_image_filename).convert()
mouse_cursor=pygame.image.load(mouse_image_filename).convert_alpha()
#加载并转换图像
whileTrue:
#游戏主循环
```

```
foreventinpygame.event.get():
ifevent.type==QUIT:
#接收到退出事件后退出程序
exit()
screen.blit(background, (0,0))
#将背景图画上去
x, y=pygame.mouse.get_pos()
#获得鼠标位置
x-=mouse_cursor.get_width()/2
y-=mouse_cursor.get_height()/2
#计算光标的左上角位置
screen.blit(mouse_cursor, (x, y))
#把光标画上去
pygame.display.update()
#刷新一下画面
```

这个程序需要两张图片，你可以在这篇文章最后的地方找到下载地址，虽然你也可以随便找两张。为了达到最佳效果，背景的 sushiplate.jpg 应要有640×480的分辨率，而光标的 fugu.png 大约应为80×80，而且要有 Alpha 通道（如果你不知道这是什么，还是下载吧……）。

注意：代码中的注释我使用的是中文，如果执行报错，可以直接删除。

游戏中我已经为每一行写了注释，另外如果打算学习，强烈建议自己动手输入一遍而不是复制粘贴！

稍微讲解一下比较重要的几个部分：

set_mode 会返回一个 Surface 对象，代表了在桌面上出现的那个窗口，三个参数第一个为元祖，代表分辨率（必须），第二个是一个标志位，具体意思见下表，如果不用什么特性，就指定0；第三个为色深。

标志位	功能
FULLSCREEN	创建一个全屏窗口
DOUBLEBUF	创建一个“双缓冲”窗口，建议在 HWSURFACE 或者 OPENGL 时使用
HWSURFACE	创建一个硬件加速的窗口，必须和 FULLSCREEN 同时使用
OPENGL	创建一个 OPENGL 渲染的窗口
RESIZABLE	创建一个可以改变大小的窗口
NOFRAME	创建一个没有边框的窗口

convert 函数是将图像数据都转化为 Surface 对象，每次加载完图像以后就应该做这件事件（事实上因为 它太常用了，如果你不写 pygame 也会帮你做）；**convert_alpha** 相比 convert，保留了 Alpha 通道信息（可以简单理解为透明的部分），这样我们的光标才可以是不规则的形状。

游戏的主循环是一个无限循环，直到用户跳出。在这个主循环里做的事情就是不停地画背景和更新光标位置，虽然背景是不动的，我们还是需要每次都画它， 否则鼠标覆盖过的位置就不能恢复正常了。**blit** 是个重要函数，第一个参数为一个 Surface 对象，第二个为左上角位置。画完以后一定记得用 **update** 更新一下，否则画面一片漆黑。

这是一个最最大概的 Pygame 程序的印象，接下来我们会学习更多深层次的东西，并且把各条语句都真正读懂。

上次我们试着写了一个最简单的 Pygame 程序并且解释了一个大概的框架，这次就 Pygame 中也是游戏中最关键（……好吧，也许并不是**最**关键，但绝对是至关重要的一项）的**事件**来展开。

此图为一个用 Pygame 开发的游戏，或许有些简陋，但是只要你有爱，什么都能出来！

理解事件

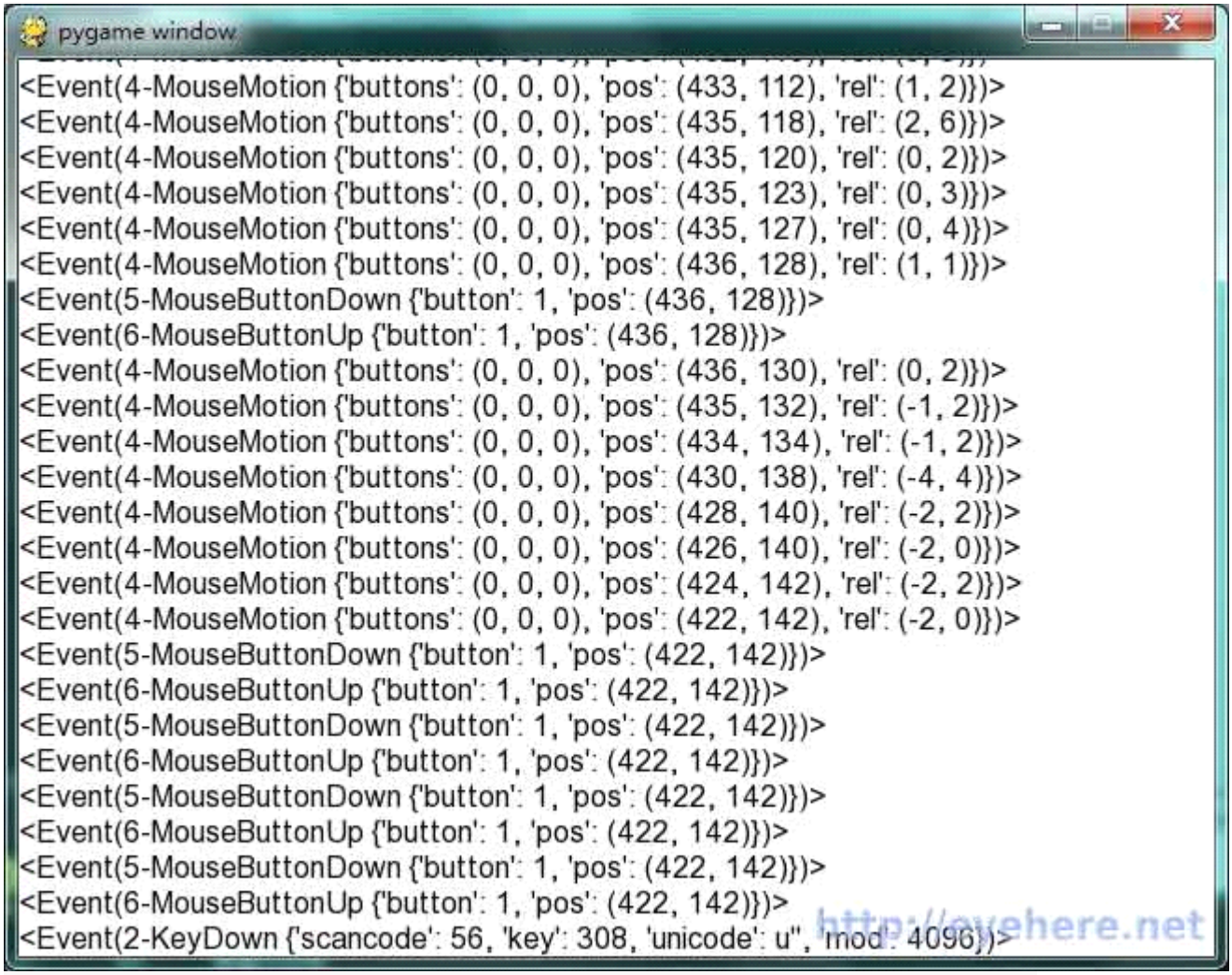
事件是什么，其实从名称来看我们就能想到些什么，而且你所想到的基本就是事件的真正意思了。我们上一个程序，会一直运行下去，直到你关闭窗口而产生了一个 QUIT 事件，Pygame 会接受用户的各种操作（比如按键盘，移动鼠标等）产生事件。事件随时可能发生，而且量也可能会很大，Pygame 的做法是把一系列的事件存放一个队列里，逐个的处理。

事件检索

上个程序中，使用了 **pygame.event.get()**来处理所有的事件，这好像打开大门让所有的人进入。如果我们使用 **pygame.event.wait()**，Pygame 就会等到发生一个事件才继续下去，就好像你在门的猫眼上盯着外面一样，来一个放一个……一般游戏中不太实用，因为游戏往往是需要动态运作的；而另外一个方法 **pygame.event.poll()**就好一些，一旦调用，它会根据现在的情形返回一个真实的事件，或者一个“什么都没有”。下表是一个常用事件集：

事件	产生途径	参数
QUIT	用户按下关闭按钮	none
ATIVEEVENT	Pygame 被激活或者隐藏	gain, state
KEYDOWN	键盘被按下	unicode, key, mod
KEYUP	键盘被放开	key, mod
MOUSEMOTION	鼠标移动	pos, rel, buttons
MOUSEBUTTONDOWN	鼠标按下	pos, button
MOUSEBUTTONUP	鼠标放开	pos, button
JOYAXISMOTION	游戏手柄(Joystick or pad)移动	joy, axis, value
JOYBALLMOTION	游戏球(Joy ball)?移动	joy, axis, value
JOYHATMOTION	游戏手柄(Joystick)?移动	joy, axis, value
JOYBUTTONDOWN	游戏手柄按下	joy, button
JOYBUTTONUP	游戏手柄放开	joy, button
VIDEORESIZE	Pygame 窗口缩放	size, w, h
VIDEOEXPOSE	Pygame 窗口部分公开(expose)?	none
USEREVENT	触发了一个用户事件	code

如果你想把这个表现在就背下来，当然我不会阻止你，但实在不是个好主意，在实际的使用中，自然而然的就会记住。我们先来写一个可以把所有方法输出的程序，它的结果是这样的。



我们这里使用了 wait()，因为这个程序在有事件发生的时候动弹就可以了。还用了 *font*模块来显示文字（后面会讲的），下面是源代码：

```
1  importpygame
2  frompygame.localsimport*
3  fromsysimportexit
4  pygame.init()
5  SCREEN_SIZE=(640,480)
6  screen=pygame.display.set_mode(SCREEN_SIZE,0,32)
7  font=pygame.font.SysFont("arial",16);
8  font_height=font.get_linesize()
9  event_text=[]
10 whileTrue:
11 event=pygame.event.wait()
12 event_text.append(str(event))
13 #获得时间的名称
14 event_text=event_text[-SCREEN_SIZE[1]/font_height:]
15 #这个切片操作保证了 event_text 里面只保留一个屏幕的文字
```

```
ifevent.type==QUIT:
exit()
screen.fill((255,255,255))
y=SCREEN_SIZE[1]-font_height
#找一个合适的起笔位置，最下面开始但是要留一行的空
fortextinreversed(event_text):
screen.blit( font.render(text,True, (0,0,0)), (0, y) )
#以后会讲
y-=font_height
#把笔提一行
pygame.display.update()
```

小贴士：

书上说，如果你把填充色的(0, 0, 0)改为(0, 255, 0)，效果会想黑客帝国的字幕雨一样，我得说，实际试一下并不太像……不过以后你完全可以写一个以假乱真甚至更酷的！

这个程序在你移动鼠标的时候产生了海量的信息，让我们知道了 Pygame 是多么的繁忙……我们第一个程序那样是调用 **pygame.mouse.get_pos()**来得到当前鼠标的位置，而现在利用事件可以直接获得！

处理鼠标事件

MOUSEMOTION 事件会在鼠标动作的时候发生，它有三个参数：

- buttons – 一个含有三个数字的元组，三个值分别代表左键、中键和右键，1就是按下了。
- pos – 就是位置了……
- rel – 代表了现在距离上次产生鼠标事件时的距离

和 MOUSEMOTION 类似的，我们还有 **MOUSEBUTTONDOWN** 和 **MOUSEBUTTONUP** 两个事

件，看名字就明白是什么意思了。很多时候，你只需要知道鼠标点下就可以了，那就可以不用上面那个比较强大（也比较复杂）的事件了。它们的参数为：

- button – 看清楚少了个 s，这个值代表了哪个按键被操作
- pos – 和上面一样

处理键盘事件

键盘和游戏手柄的事件比较类似，为 **KEYDOWN** 和 **KEYUP**，下面有一个例子来演示使用方向键移动一些东西。

```
background_image_filename='sushiplate.jpg'
importpygame
frompygame.localsimport*
fromsysimportexit
pygame.init()
screen=pygame.display.set_mode((640,480),0,32)
background=pygame.image.load(background_image_filename).convert()
x, y=0,0
move_x, move_y=0,0
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
exit()
ifevent.type==KEYDOWN:
#键盘有按下？
ifevent.key==K_LEFT:
#按下的是左方向键的话，把 x 坐标减一
move_x=-1
elifevent.key==K_RIGHT:
#右方向键则加一
move_x=1
elifevent.key==K_UP:
#类似了
move_y=-1
elifevent.key==K_DOWN:
move_y=1
elifevent.type==KEYUP:
#如果用户放开了键盘，图就不要动了
move_x=0
move_y=0
```

```
#计算出新的坐标
x+=move_x
y+=move_y
screen.fill((0,0,0))
screen.blit(background, (x,y))
#在新的位置上画图
pygame.display.update()
```

当我们运行这个程序的时候，按下方向键就可以把背景图移动，但是等等！为什么我只能按一下动一下啊……太不好试了吧？！用脚掌考虑下就应该按着就一直动下去才是啊 ？Pygame 这么垃圾么……哦，真是抱歉上面的代码有点小 bug，但是真的很小，你都不需要更改代码本身，只要**改一下缩进**就可以了，你可以发现么？Python 本身是缩进编排来表现层次，有些时候可能会出现一点小麻烦，要我们自己注意才可以。

KEYDOWN 和 KEYUP 的参数描述如下：

- key – 按下或者放开的键值，是一个数字，估计地球上很少有人可以记住，所以 Pygame 中你可以使用 **K_xxx** 来表示，比如字母 a 就是 K_a，还有 **K_SPACE** 和 **K_RETURN** 等。
- mod – 包含了组合键信息，如果 mod & **KMOD_CTRL** 是真的话，表示用户同时按下了 Ctrl 键。类似的还有 **KMOD_SHIFT**，**KMOD_ALT**。
- unicode – 代表了按下键的 Unicode 值，这个有点不好理解，真正说清楚又太麻烦，游戏中也不太常用，说明暂时省略，什么时候需要再讲吧。

事件过滤

并不是所有的事件都需要处理的，就好像不是所有登门造访的人都是我们欢迎的一样。比如，俄罗斯方块就无视你的鼠标，而在游戏场景切换的时候，你按什么都是徒劳的。我们应该有一个方法来过滤

掉一些我们不感兴趣的事件（当然我们可以不处理这些没兴趣的事件，但最好的方法还是让它们根本不进入我们的事件队列，就好像在门上贴着“XXX免进”一样），我们使用**pygame.event.set_blocked(事件名)**来完成。如果有好多事件需要过滤，可以传递一个列表，比如pygame.event.set_blocked([KEYDOWN, KEYUP])，如果你设置参数None，那么所有的事件有被打开了。与之相对的，我们使用**pygame.event.set_allowed()**来设定允许的事件。

产生事件

通常玩家做什么，Pygame就产生对应的事件就可以了，不过有的时候我们需要模拟出一些事件来，比如录像回放的时候，我们就要把用户的操作再现一遍。

为了产生事件，必须先造一个出来，然后再传递它：

```
my_event=pygame.event.Event(KEYDOWN, key=K_SPACE, mod=0,unicode=u' ')
#你也可以像下面这样写，看起来比较清晰（但字变多了……）
my_event=pygame.event.Event(KEYDOWN, {"key":K_SPACE,"mod":0,"unicode":u' '})
pygame.event.post(my_event)

```

你甚至可以产生一个完全自定义的全新事件，有些高级的话题，暂时不详细说，仅用代码演示一下：

```
CATONKEYBOARD=USEREVENT+1
my_event=pygame.event.Event(CATONKEYBOARD, message="Bad cat!")
pgame.event.post(my_event)
#然后获得它
foreventinpygame.event.get():
ifevent.type==CATONKEYBOARD:
printevent.message

```

这次的内容很多，又很重要，一遍看下来云里雾里或者看的时候明白看完了全忘了什么的估计很多，慢慢学习吧~~多看看动手写写，其实都很简单。

下次讲解显示的部分。

OK，到该讲显示的时候了。没人可以否定好的画面是一款游戏吸引人最直接最诱人的因素，虽说滥画面高游戏度的作品也有，但优秀的画面无疑是一张过硬的通行证，可以让你争取到更多的机会。

其实上两回也已经打开过显示了，不过没有特别说明而已，pygame.display.set_mode(xxx)就是创建一个游戏窗口，也就是显示的意思。

全屏显示

我们在第一个程序里使用了如下的语句

```
screen=pygame.display.set_mode((640,480),0,32)
```

也讲述了各个参数的意思，当我们把第二个参数设置为 FULLSCREEN 时，就能得到一个全屏窗口了

```
screen=pygame.display.set_mode((640,480), FULLSCREEN,32)
```

注意：如果你的程序有什么问题，很可能进入了全屏模式就不太容易退出来了，所以最好先用窗口模式调试好，再改为全屏模式。

在全屏模式下，显卡可能就切换了一种模式，你可以用如下代码获得您的机器支持的显示模式：

```
>>>importpygame
>>> pygame.init()
```

```
>>> pygame.display.list_modes()
```

看一下一个实例：

```


background_image_filename='sushiplate.jpg'
importpygame
frompygame.localsimport*
fromsysimportexit
pygame.init()
screen=pygame.display.set_mode((640,480),0,32)
background=pygame.image.load(background_image_filename).convert()
Fullscreen=False
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
exit()
ifevent.type==KEYDOWN:
ifevent.key==K_f:
Fullscreen=notFullscreen
ifFullscreen:
screen=pygame.display.set_mode((640,480), FULLSCREEN,32)
else:
screen=pygame.display.set_mode((640,480),0,32)
screen.blit(background, (0,0))
pygame.display.update()



```

运行这个程序，默认还是窗口的，按“f”，显示模式会在窗口和全屏之间切换。程序也没有什么难度，应该都能看明白。

可变尺寸的显示

虽然一般的程序窗口都能拖边框来改变大小，pygame 的默认显示窗口是不行的，而事实上，很多游戏确实也不能改变显示窗口的大小，我们可以使用一个参数来改变这个默认行为。

```
background_image_filename='sushiplate.jpg'
importpygame
frompygame.localsimport*
```

```
fromsysimportexit
SCREEN_SIZE=(640,480)
pygame.init()
screen=pygame.display.set_mode(SCREEN_SIZE, RESIZABLE,32)
background=pygame.image.load(background_image_filename).convert()
whileTrue:
event=pygame.event.wait()
ifevent.type==QUIT:
exit()
ifevent.type==VIDEORESIZE:
SCREEN_SIZE=event.size
screen=pygame.display.set_mode(SCREEN_SIZE, RESIZABLE,32)
pygame.display.set_caption("Window resized to "+str(event.size))
screen_width, screen_height=SCREEN_SIZE
# 这里需要重新填满窗口
foryinrange(0, screen_height, background.get_height()):
forxinrange(0, screen_width, background.get_width()):
screen.blit(background, (x, y))
pygame.display.update()
```

当你更改大小的时候，后端控制台会显示出新的尺寸，这里我们学习到一个新的事件 **VIDEORESIZE**，

它包含如下内容：

- size — 一个二维元组，值为更改后的窗口尺寸，size[0]为宽，size[1]为高
- w — 宽
- h — 一目了然，高；之所以多出这两个，无非是为了方便

注意:在我的 Windows 7 64bit 上运行的时候，一改变窗口大小就非法退出；在 Linux 机器上很正常，

应该是系统的兼容性问题（Pygame 还只支持32位，，不过想来平时都不会更改游戏窗口大小，问题

不大。

至于无边框的窗口等，看一看本教程的第一篇就能知道了，不再赘述。

其他、复合模式

我们还有一些其他的显示模式，但未必所有的操作系统都支持(放心 windows、各种比较流行的 Linux 发行版都是没问题的)，一般来说窗口就用0全屏就用 FULLSCREEN，这两个总是 OK 的。

如果你想创建一个硬件显示(surface 会存放在显存里，从而有着更高的速度)，你必须和全屏一起使用：

```
screen=pygame.display.set_mode(SCREEN_SIZE, HWSURFACE | FULLSCREEN,32)
```

当然你完全可以把双缓冲(更快)DOUBLEBUF 也加上，这就是一个很棒的游戏显示了，不过记得你要使用 **pygame.display.flip()**来刷新显示。pygame.display.update()是将数据画到前面显示，而这个是交替显示的意思。

稍微说一下双缓冲的意思，可以做一个比喻：我的任务就是出黑板报，如果只有一块黑板，那我得不停的写，全部写完了稍微 Show 一下就要擦掉重写，这样一来别人看的基本都是我在写黑板报的过程，看到的都是不完整的黑板报；如果我有两块黑板，那么可以挂一块给别人看，我自己在底下写另一块，写好了把原来的换下来换上新的，这样一来别人基本总是看到完整的内容了。双缓冲就是这样维护两个显示区域，快速的往屏幕上换内容，而不是每次都慢慢地重画。

还有 OPENGL 模式，这是一个得到广泛应用的3D 加速显示模式。不过一旦使用了这个模式，pygame 中的2D 图像函数就不能用了，我们会在以后讲详细的内容。

这次的东西不是很多，基本就是讲了一个显示参数，如果基础比较好，一看就明白了。不过还是建议实际的输入写一下巩固认识。下一回讲字体模块(游戏没图可以，没字咋整？)~尽请期待

使用字体模块

就像上一次说的，一个游戏，再怎么寒碜也得有文字，俄罗斯方块还有个记分数的呢；印象中没有文字的电子游戏只有电脑刚刚诞生的那种打乒乓的了。Pygame 可以直接调用系统字体，或者也可以使用 TTF 字体，稍有点电脑知识的都知道这是什么。为了使用字体，你得先创建一个 Font 对象，对于

系统自带的字体：

```
my_font=pygame.font.SysFont("arial",16)
```

第一个参数是字体名，第二个自然就是大小，一般来说“Arial”字体在很多系统都是存在的，如果找不到的话，就会使用一个默认的字体，这个默认的字体和每个操作系统相关，你也可以使用**pygame.font.get_fonts()**来获得当前系统所有可用字体。还有一个更好的方法的，使用 TTF 的方法：

```
my_font=pygame.font.Font("my_font.ttf",16)
```

这个语句使用了一个叫做“my_font.ttf”，这个方法之所以好是因为你可以把字体文件随游戏一起分发，避免用户机器上没有需要的字体。。一旦你创建了一个 font 对象，你就可以使用 render 方法来写字了，然后就能 blit 到屏幕上：

```
text_surface=my_font.render("Pygame is cool!",True, (0,0,0), (255,255,255))
```

第一个参数是写的文字；第二个参数是个布尔值，以为这是否开启抗锯齿，就是说 True 的话字体会比较平滑，不过相应的速度有一点点影响；第三个参数是字体的颜色；第四个是背景色，如果你想没有背景色（也就是透明），那么可以不加这第四个参数。

下面是一个小例子演示下文字的使用，不过并不是显示在屏幕上，而是存成一个图片文件

```
my_name="Will McGugan"
importpygame
pygame.init()
my_font=pygame.font.SysFont("arial",64)
name_surface=my_font.render(my_name,True, (0,0,0), (255,255,255))
pygame.image.save(name_surface,"name.png")
```

追加说明一下如何显示中文，这在原书可是没有的哦：）简单来说，首先你得用一个可以使用中文的字体，宋体、黑体什么的，或者你直接用中文 TTF 文件，然后文字使用 unicode，即 u”中文的文字”这种，最后不要忘了源文件里加上一句关于文件编码的“魔法注释”，具体的可以查一下 Python 的

编码方面的文章。举一个这样的例子：

```
# -*- coding: utf-8 -*-
# 记住上面这行是必须的，而且保存文件的编码要一致！
importpygame
frompygame.localsimport*
fromsysimportexit
pygame.init()
screen=pygame.display.set_mode((640,480),0,32)
#font = pygame.font.SysFont("宋体", 40)
#上句在 Linux 可行，在我的 Windows 7 64bit 上不行，XP 不知道行不行
#font = pygame.font.SysFont("simsunnsimsun", 40)
#用 get_fonts()查看后看到了这个字体名，在我的机器上可以正常显示了
font=pygame.font.Font("simsun.ttc",40)
#这句话总是可以的，所以还是 TTF 文件保险啊
text_surface=font.render(u"你好",True, (0,0,255))
x=0
y=(480-text_surface.get_height())/2
background=pygame.image.load("sushiplate.jpg").convert()
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
exit()
screen.blit(background, (0,0))
x-=2# 文字滚动太快的话，改改这个数字
ifx <-text_surface.get_width():
x=640-text_surface.get_width()
screen.blit(text_surface, (x, y))
pygame.display.update()
```

Pygame 的错误处理

程序总会出错的，比如当内存用尽的时候 Pygame 就无法再加载图片，或者文件根本就不存在。再比如下例：

```
>>>importpygame
>>> screen=pygame.display.set_mode((640,-1))
---------------------------------
Traceback (most recent call last):
File"<interactive input>", line1,in?
pygame.error: Cannotset0sized display mode
---------------------------------
```

对付这种错误一个比较好的方法：

```
try:
screen=pygame.display.set_mode(SCREEN_SIZE)
exceptpygame.error, e:
print"Can't create the display :-("
printe
exit()
```

其实就是 Python 的标准的错误捕捉方法就是了，实际的游戏（或者程序）中，错误捕捉实在太重要了，如果你写过比较大的应用，应该不用我来说明这一点，Pygame 中也是一样的。

Pygame 的基础就到这里，后面我们会进行一些高级的介绍，下一次的话，就开始讲画东西了~

这次开始是真正的游戏编程，以前都是基础的基础啊。

电脑游戏总是倾向于图像化的，尽量的要看得到听得到（现在的技术基本还局限于这两个感官），游戏开发者会花无数的力气在图像上，提升图像效果是游戏开发永恒的话题。这几次主要讲述游戏中的视觉。

像素的威力

凑近显示器，你能看到图像是由一个一个点构成，这就是像素。至于屏幕分辨率的意义，也就不用多说了吧，一个1280×1024的显示器，有着1310720个像素，一般的32为 RGB 系统，每个像素可以显示16.7百万种颜色（可以看我的另一篇一张白纸可以承载多少重的文章），我们可以写一个小程序来显示这么多的颜色~

```
importpygame
pygame.init()
screen=pygame.display.set_mode((640,480))
all_colors=pygame.Surface((4096,4096), depth=24)
forrinxrange(256):
printr+1,"out of 256"
x=(r&15)*256
y=(r>>4)*256
forginxrange(256):
forbinxrange(256):
all_colors.set_at((x+g, y+b), (r, g, b))
pygame.image.save(all_colors,"allcolors.bmp")
```

运行可能有些慢，你应该等生成 bmp 图像文件，打开看看效果吧（其实就是我刚刚提到的博文里的

图片）。

色彩的威力

色彩是一个很有趣的话题，比如把蓝色和黄色混合产生绿色，事实上你可以用红黄蓝混合出所有的颜色（光学三原色），电脑屏幕上的三原色是红绿蓝（RGB），要想更深刻的理解这个东西，你得学习一下（就看看李涛的 PhotoShop 讲座吧，VeryCD 上有下的，讲的还是很清楚的）~

稍有点经验的图像设计者应该看到 RGB 的数值就能想象出大概的颜色，我们来用一个 Python 脚本加强这个认识。

```
#!/usr/bin/env python
importpygame
frompygame.localsimport*
fromsysimportexit
pygame.init()
screen=pygame.display.set_mode((640,480),0,32)
defcreate_scales(height):
red_scale_surface=pygame.surface.Surface((640, height))
green_scale_surface=pygame.surface.Surface((640, height))
blue_scale_surface=pygame.surface.Surface((640, height))
forxinrange(640):
c=int((x/640.)*255.)
red=(c,0,0)
green=(0, c,0)
blue=(0,0, c)
line_rect=Rect(x,0,1, height)
pygame.draw.rect(red_scale_surface, red, line_rect)
pygame.draw.rect(green_scale_surface, green, line_rect)
pygame.draw.rect(blue_scale_surface, blue, line_rect)
returnred_scale_surface, green_scale_surface, blue_scale_surface
red_scale, green_scale, blue_scale=create_scales(80)
color=[127,127,127]
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
exit()
screen.fill((0,0,0))
screen.blit(red_scale, (0,00))
screen.blit(green_scale, (0,80))
screen.blit(blue_scale, (0,160))
```

```
x, y=pygame.mouse.get_pos()
ifpygame.mouse.get_pressed()[0]:
forcomponentinrange(3):
ify > component*80andy < (component+1)*80:
color[component]=int((x/639.)*255.)
pygame.display.set_caption("PyGame Color Test - "+str(tuple(color)))
forcomponentinrange(3):
pos=(int((color[component]/255.)*639), component*80+40)
pygame.draw.circle(screen, (255,255,255), pos,20)
pygame.draw.rect(screen,tuple(color), (0,240,640,240))
pygame.display.update()
```

这个程序稍稍有点难度了，而且用到了一些没讲到的知识（pygame.draw），我们以后会介绍，现在无所谓。在这个例子里，你可以用鼠标移动三个白点，代表了三原色的量，下面就是不同混合得到的结果，在标题上你可以看到 RGB 三个数值。

当我们有了一个颜色，比如说一颗流星划过天际，那么那个时候它是个“火球般的橘黄色”，不过一旦它着地了，它就会灭掉，慢慢变暗，如何能找到比这个“火球般的橘黄色”更暗的颜色？

颜色的缩放

“缩放颜色”并不是一种合适的说法，它的准确意义就是上面所说的把颜色变亮或者变暗。一般来说，把颜色的 RGB 每一个数值乘以一个小于1的正小数，颜色看起来就会变暗了（记住 RGB 都是整数所以可能需要取整一下）。我们很容易可以写一个缩放颜色的函数出来，我就不赘述了。

很自然的可以想到，如果乘以一个大于1的数，颜色就会变亮，不过同样要记住每个数值最多255，所以一旦超过，你得把它归为255！使用 Python 的内置函数 **min**，你可以方便的做到这事情，也不多说了。如果你乘的数字偏大，颜色很容易就为变成纯白色，就失去了原来的色调。而且 RGB 也不可能是负数，所以谨慎选择你的缩放系数！

颜色的混合

很多时候我们还需要混合颜色，比如一个僵尸在路过一个火山熔岩坑的时候，它会由绿色变成橙红色，再变为正常的绿色，这个过程必须表现的很平滑，这时候我们就需要混合颜色。

我们用一种叫做“**线性插值(linear interpolation)**”的方法来做这件事情。为了找到两种颜色的中间色，我们将这第二种颜色与第一种颜色的差乘以一个0~1之间的小数，然后再加上第一种颜色就行了。如果这个数为0，结果就完全是第一种颜色；是1，结果就只剩下第二种颜色；中间的小数则会皆有两者的特色。

```
#!/usr/bin/env python
importpygame
frompygame.localsimport*
fromsysimportexit
pygame.init()
screen=pygame.display.set_mode((640,480),0,32)
color1=(221,99,20)
color2=(96,130,51)
factor=0.
defblend_color(color1, color2, blend_factor):
r1, g1, b1=color1
r2, g2, b2=color2
r=r1+(r2-r1)*blend_factor
g=g1+(g2-g1)*blend_factor
b=b1+(b2-b1)*blend_factor
returnint(r),int(g),int(b)
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
exit()
screen.fill((255,255,255))
tri=[ (0,120), (639,100), (639,140) ]
```

```
pygame.draw.polygon(screen, (0,255,0), tri)
pygame.draw.circle(screen, (0,0,0), (int(factor*639.0),120),10)
x, y=pygame.mouse.get_pos()
ifpygame.mouse.get_pressed()[0]:
factor=x/639.0
pygame.display.set_caption("Pygame Color Blend Test - %.3f"%factor)
color=blend_color(color1, color2 , factor)
pygame.draw.rect(screen, color, (0,240,640,240))
pygame.display.update()
```

在这里例子里，移动小球你能看到下方的颜色在“火球橙”和“僵尸绿”之间渐变，更改代码里的color1和 color2，你能看到任意两种颜色渐变的过程！

今天主要说明了像素和色彩，很简单，但确实是要点，多写写程序试试，好好理解理解吧！

掌握了小小的像素，我们可以使用更加复杂一点的东西了，对，就是**图像**，无数的像素的集合~还记得上次我们为了生成的一张图片，花了无数时间，还好一般游戏不会在游戏的过程中动态生成图像，都是将画好的作为资源封装到游戏中。对2D 游戏，图像可能就是一些背景、角色等，而3D 游戏则往往是大量的贴图。

虽然是基础，这里还是要罗嗦一下，之前说的 RBG 图像，在游戏中我们往往使用 RGBA 图像，这个 A 是 alpha，也就是表示透明度的部分，值也是0~255，0代表完全透明，255是完全不透明，而像100这样的数字，代表部分透明。你可以使用多种软件创建含有 Alpha 通道的图片，具体的网上查查吧……

这个世界上有很多存储图像的方式（也就是有很多图片格式），比如 JPEG、PNG 等，Pygmae 都能

很好的支持，具体支持的格式如下：

- JPEG（Join Photograhpic Exper Group)，极为常用，一般后缀名为.jpg 或者.jpeg。数码相机、网上的图片基本都是这种格式。这是一种有损压缩方式，尽管对图片质量有些损坏，但对于减小文件尺寸非常棒。优点很多只是不支持透明。
- PNG（Portable Network Graphics）将会大行其道的一种格式，支持透明，无损压缩。对于网页设计，软件界面设计等等都是非常棒的选择！
- GIF 网上使用的很多，支持透明和动画，只是只能有256种颜色，软件和游戏中使用很少
- BMP Windows 上的标准图像格式，无压缩，质量很高但尺寸很大，一般不使用
- PCX
- TGA
- TIF
- LBM, PBM
- XPM

使用 Surface 对象

对于 Pygame 而已 加载图片就是 pygame.image.load 给它一个文件名然后就还给你一个 surface 对象。尽管读入的图像格式各不相同，surface 对象隐藏了这些不同。你可以对一个 Surface 对象进行涂画、变形、复制等各种操作。事实上，屏幕也只是一个 surface，pygame.display.set_mode 就返回了一个屏幕 surface 对象。

创建 Surfaces 对象

一种方法就是刚刚说的 pygame.image.load，这个 surface 有着和图像相同的尺寸和颜色；另外一种方法是指定尺寸创建一个空的 surface，下面的语句创建一个256×256像素的 surface：

```
1 bland_surface=pygame.Surface((256,256))
```

如果不指定尺寸，那么就创建一个和屏幕一样大小的。

你还有两个参数可选，第一个是 flags：

- HWSURFACE – 类似于前面讲的，更快！不过最好不设定，Pygmae 可以自己优化。
- SRCALPHA – 有 Alpha 通道的 surface，如果你需要透明，就要这个选项。这个选项的使用需要第二个参数为32~

第二个参数是 depth，和 pygame.display.set_mode 中的一样，你可以不设定，Pygame 会自动设的和 display 一致。不过如果你使用了 SRCALPHA，还是设为32吧：

```
1 bland_alpha_surface=pygame.Surface((256,256), flags=SRCALPHA, depth=32)
```

转换 Surfaces

通常你不用在意 surface 里的具体内容，不过也许需要把这些 surface 转换一下以获得更高的性能，还记得一开始的程序中的两句话吗：

```
background=pygame.image.load(background_image_filename).convert()
mouse_cursor=pygame.image.load(mouse_image_filename).convert_alpha()
```

第一句是普通的转换，相同于 display；第二句是带 alpha 通道的转换。如果你给 convert 或者 conver_alpha 一个 surface 对象作为参数，那么这个会被作为目标来转换。

矩形对象(Rectangle Objects)

一般来说在制定一个区域的时候，矩形是必须的，比如在屏幕的一部分画东西。在 pygame 中矩形对象极为常用，它的指定方法可以用一个四元素的元组，或者两个二元素的元组，前两个数为左上坐标，后两位为右下坐标。

Pygame 中有一个 Rect 类，用来存储和处理矩形对象（包含在 pygame.locals 中，所以如果你写了 from pygame.locals import *就可以直接用这个对象了），比如：

```
my_rect1=(100,100,200,150)
my_rect2=((100,100), (200,150))
#上两种为基础方法，表示的矩形也是一样的
my_rect3=Rect(100,100,200,150)
my_rect4=Rect((100,100), (200,150))
```

一旦有了 Rect 对象，我们就可以对其做很多操作，比如调整位置和大小，判断一个点是否在其中等等。以后会慢慢接触到，求知欲旺盛的可以在 http://www.pygame.org/docs/ref/rect.html 中找到 Rect 的详细信息。

剪裁(Clipping)

通常游戏的时候你只需要绘制屏幕的一部分。比如魔兽上面是菜单，下面是操作面板，中间的小兵和英雄打的不可开交时候，上下的部分也是保持相对不动的。为了实现这一点，surface 就有了一种叫*裁剪区域（clipping area)*的东西，也是一个矩形，定义了哪部分会被绘制，也就是说一旦定义了这个区域，那么只有这个区域内的像素会被修改，其他的位置保持不变，默认情况下，这个区域是所有地方。我们可以使用 set_clip 来设定，使用 get_clip 来获得这个区域。

下面几句话演示了如何使用这个技术来绘制不同的区域：

```
screen.set_clip(0,400,200,600)
draw_map()
#在左下角画地图
screen.set_clip(0,0,800,60)
draw_panel()
#在上方画菜单面板
```

子表面(Subsurfaces)

Subsurface 就是在一个 Surface 中再提取一个 Surface，记住当你往 Subsurface 上画东西的时候，同时也向父表面上操作。这可以用来绘制图形文字，尽管 pygame.font 可以用来写很不错的字，但只是单色，游戏可能需要更丰富的表现，这时候你可以把每个字母（中文的话有些吃力了）各自做成一个图片，不过更好的方法是在一张图片上画满所有的字母。把整张图读入，然后再用 Subsurface 把字母一个一个“抠”出来，就像下面这样：

```
my_font_image=Pygame.load("font.png")
letters=[]
letters["a"]=my_font_image.subsurface((0,0), (80,80))
letters["b"]=my_font_image.subsurface((80,0), (80,80))
```

填充 Surface

填充有时候可以作为一种清屏的操作，把整个 surface 填上一种颜色：

```
screen.fill((0,0,0))
```

同样可以提供一个矩形来制定填充哪个部分（这也可以作为一种画矩形的方法）。

设置 Surface 的像素

我们能对 Surface 做的最基本的操作就是设置一个像素的色彩了，虽然我们基本不会这么做，但还是要了解。set_at 方法可以做到这一点，它的参数是坐标和颜色，下面的小脚本会随机的在屏幕上画点：

```
importpygame
frompygame.localsimport*
fromsysimportexit
fromrandomimportrandint
pygame.init()
screen=pygame.display.set_mode((640,480),0,32)
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
exit()
rand_col=(randint(0,255), randint(0,255), randint(0,255))
#screen.lock() #很快你就会知道这两句 lock 和 unlock 的意思了
for_inxrange(100):
rand_pos=(randint(0,639), randint(0,479))
screen.set_at(rand_pos, rand_col)
```

```
16  #screen.unlock()
17  pygame.display.update()
18
19
20
21
```

获得 Surface 上的像素

set_at 的兄弟 get_at 可以帮我们做这件事，它接受一个坐标返回指定坐标点上的颜色。不过记住 get_at 在对 hardware surface 操作的时候很慢，而全屏的时候总是 hardware 的，所以慎用这个方法！

锁定 Surface

当 Pygame 往 surface 上画东西的时候，首先会把 surface 锁住，以保证不会有其它的进程来干扰，画完之后再解锁。锁和解锁时自动发生的，所以有时候可能不那么有效率，比如上面的例子，每次画100个点，那么就得锁解锁100次，现在我们把两句注释去掉，再执行看看是不是更快了（好吧，其实我没感觉出来，因为现在的机器性能都不错，这么点的差异还不太感觉的出来。不过请相信我~复杂的情况下会影响效率的）?

当你手动加锁的时候，一定不要忘记解锁，否则 pygame 有可能会失去响应。虽然上面的例子可能没问题，但是隐含的 bug 是我们一定要避免的事情。

Blitting

blit 的的中文翻译给人摸不着头脑的感觉，可以译为位块传送（bit block transfer），其意义是将一个平面的一部分或全部图象整块从这个平面复制到另一个平面，下面还是直接使用英文。

blit 是对表面做的最多的操作，我们在前面的程序中已经多次用到，不多说了；blit 的还有一种用法，往往用在对动画的表现上，比如下例通过对 frame_no 的值的改变，我们可以把不同的帧（同一副图的不同位置）画到屏幕上：

```
1 screen.blit(ogre, (300,200), (100*frame_no,0,100,100))
```

这次东西真是不少，打完脖子都酸了……

很多以前的程序中已经出现，看完这部分才能算是真正了解。图像是游戏至关重要的一部分，值得多花时间，下一次讲解绘制图形~

们上一个章节使用了 pygame.draw 中的一些函数，这个模块的作用是在屏幕上绘制各种图形。事实上，你可以不加载任何图片，只是要这些图形来制作一个游戏（经典游戏 Asteroids 便是，这里有一个 HTML5写就的例子。好像访问不了？搜个 Flash 版吧，多得很）。

注：该图与该文无任何联系（最近在玩神鬼寓言3，感觉还是不错的~）

pygame.draw 中函数的第一个参数总是一个 surface，然后是颜色，再后会是一系列的坐标等。稍有些计算机绘图经验的人就会知道，计算机里的坐标，(0，0)代表左上角。而返回值是一个 Rect 对象，包含了绘制的领域，这样你就可以很方便的更新那个部分了。

函数	作用
rect	绘制矩形
polygon	绘制多边形（三个及三个以上的边）
circle	绘制圆
ellipse	绘制椭圆
arc	绘制圆弧
line	绘制线
lines	绘制一系列的线
aaline	绘制一根平滑的线
aalines	绘制一系列平滑的线

我们下面一个一个详细说明。

pygame.draw.rect

用法：pygame.draw.rect(Surface, color, Rect, width=0)

pygame.draw.rect 在 surface 上画一个矩形，除了 surface 和 color，rect 接受一个矩形的坐标和线宽参数，如果线宽是0或省略，则填充。我们有一个另外的方法来画矩形——fill 方法，如果你还记得的话。事实上 fill 可能还会快一点点，因为 fill 由显卡来完成。

pygame.draw.polygon

用法：pygame.draw.polygon(Surface, color, pointlist, width=0)

polygon 就是多边形，用法类似 rect，第一、第二、第四的参数都是相同的，只不过 polygon 会接受一系列坐标的列表，代表了各个顶点。

pygame.draw.circle

用法：pygame.draw.circle(Surface, color, pos, radius, width=0)

很简单，画一个圆。与其他不同的是，它接收一个圆心坐标和半径参数。

pygame.draw.ellipse

用法：pygame.draw.ellipse(Surface, color, Rect, width=0)

你可以把一个 ellipse 想象成一个被压扁的圆，事实上，它是可以被一个矩形装起来的。

pygame.draw.ellipse 的第三个参数就是这个椭圆的外接矩形。

pygame.draw.arc

用法：pygame.draw.arc(Surface, color, Rect, start_angle, stop_angle, width=1)

arc 是椭圆的一部分，所以它的参数也就比椭圆多一点。但它是不封闭的，因此没有 fill 方法。

start_angle 和 stop_angle 为开始和结束的角度。

pygame.draw.line

用法：pygame.draw.line(Surface, color, start_pos, end_pos, width=1)

我相信所有的人都能看明白。

pygame.draw.lines

用法：pygame.draw.lines(Surface, color, closed, pointlist, width=1)

closed 是一个布尔变量，指明是否需要多画一条线来使这些线条闭合（感觉就和 polygone 一样了），

pointlist 是一个点的数组。

上面的表中我们还有 aaline 和 aalines，玩游戏的都知道开出“抗锯齿（antialiasing）”效果会让画面更好看一些，模型的边就不会是锯齿形的了，这两个方法就是在画线的时候做这事情的，参数和上面一样，省略。

我们用一个混杂的例子来演示一下上面的各个方法：

```
#!/usr/bin/env python
importpygame
frompygame.localsimport*
fromsysimportexit
fromrandomimport*
frommathimportpi
pygame.init()
screen=pygame.display.set_mode((640,480),0,32)
points=[]
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
exit()
ifevent.type==KEYDOWN:
# 按任意键可以清屏并把点回复到原始状态
points=[]
screen.fill((255,255,255))
ifevent.type==MOUSEBUTTONDOWN:
screen.fill((255,255,255))
# 画随机矩形
rc=(randint(0,255), randint(0,255), randint(0,255))
rp=(randint(0,639), randint(0,479))
rs=(639-randint(rp[0],639),479-randint(rp[1],479))
pygame.draw.rect(screen, rc, Rect(rp, rs))
# 画随机圆形
rc=(randint(0,255), randint(0,255), randint(0,255))
rp=(randint(0,639), randint(0,479))
rr=randint(1,200)
pygame.draw.circle(screen, rc, rp, rr)
# 获得当前鼠标点击位置
x, y=pygame.mouse.get_pos()
points.append((x, y))
# 根据点击位置画弧线
angle=(x/639.)*pi*2.
pygame.draw.arc(screen, (0,0,0), (0,0,639,479),0, angle,3)
# 根据点击位置画椭圆
pygame.draw.ellipse(screen, (0,255,0), (0,0, x, y))
# 从左上和右下画两根线连接到点击位置
pygame.draw.line(screen, (0,0,255), (0,0), (x, y))
pygame.draw.line(screen, (255,0,0), (640,480), (x, y))
# 画点击轨迹图
iflen(points) >1:
pygame.draw.lines(screen, (155,155,0),False, points,2)
# 和轨迹图基本一样，只不过是闭合的，因为会覆盖，所以这里注释了
```

```
#if len(points) >= 3:
# pygame.draw.polygon(screen, (0, 155, 155), points, 2)
# 把每个点画明显一点
forpinpoints:
pygame.draw.circle(screen, (155,155,155), p,3)
pygame.display.update()
```

运行这个程序，在上面点鼠标就会有图形出来了；按任意键可以重新开始。另外这个程序只是各个命令的堆砌，并不见得是一个好的程序代码。

到这次为止，文字、颜色、图像、图形都讲好了，静态显示的部分都差不多了。然而多彩的游戏中只有静态的东西实在太让人寒心了（GalGame大多如此），下次开始我们学习游戏中的动画制作。

是时候让我们的游戏活泼起来了。电脑游戏和桌面游戏的一个巨大差别，想来就是这个“动”。伟大的哲学家们告诉我们，“运动是绝对的，静止时相对的”，同样的在游戏中，只有活动起来，游戏才会拥有生命，否则和看连环画有什么差别呢？

这几章讲述的东西需要一些线性代数的知识，好吧有些夸张，如果你不明白，完全没关系，高中物理的知识就绝对足够了（或者说嫌多了）！

现实生活中的物体，运动起来总是按照某种规律的（去问问牛顿就知道了），而游戏中，有些动作就可以非常的不靠谱，比如吃豆人，大嘴巴永远以恒定的速度前进，可以瞬间转身或停止，要知道，这可是逆天的行为……现在的游戏中，制作者总是尽量的把运动做的和现实贴近（尤其是赛车游戏等），一辆车的运动，可能是上百种力同时作用的结果。不过幸好，我们只要知道一些基础的东西，很多运动和力的计算，都有现成的代码供我们使用。

理解帧率

这是一个被说烂了的词，FPS（Frame Per Second）是游戏和硬件间较量的永恒话题，我也不想多插话了，相信玩游戏的朋友都知道。

只是记住几个常用的量：一般的电视画面是24FPS；30FPS 基本可以给玩家提供流程的体验了；LCD 的话，60FPS 是常用的刷新率，所以你的游戏的帧率再高也就没什么意义了；而绝大多数地球人都无法分辨70FPS 以上的画面了！

直线运动

我们先来看一下初中一开始就学习的直线运动，我们让一开始的程序中出现的那条鱼自己动起来~

```
background_image_filename='sushiplate.jpg'
sprite_image_filename='fugu.png'
```

```
importpygame
frompygame.localsimport*
fromsysimportexit
pygame.init()
screen=pygame.display.set_mode((640,480),0,32)
background=pygame.image.load(background_image_filename).convert()
sprite=pygame.image.load(sprite_image_filename)
# sprite 的起始 x 坐标
x=0.
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
exit()
screen.blit(background, (0,0))
screen.blit(sprite, (x,100))
x+=10.#如果你的机器性能太好以至于看不清，可以把这个数字改小一些
# 如果移动出屏幕了，就搬到开始位置继续
ifx >640.:
x=0.
pygame.display.update()
```

我想你应该需要调节一下“x += 10”来让这条鱼游的自然一点，不过，这个动画的帧率是多少的？在这个情形下，动画很简单，所以应该会很快；而有些时候动画元素很多，速度就会慢下来。这可不是我们想看到的！

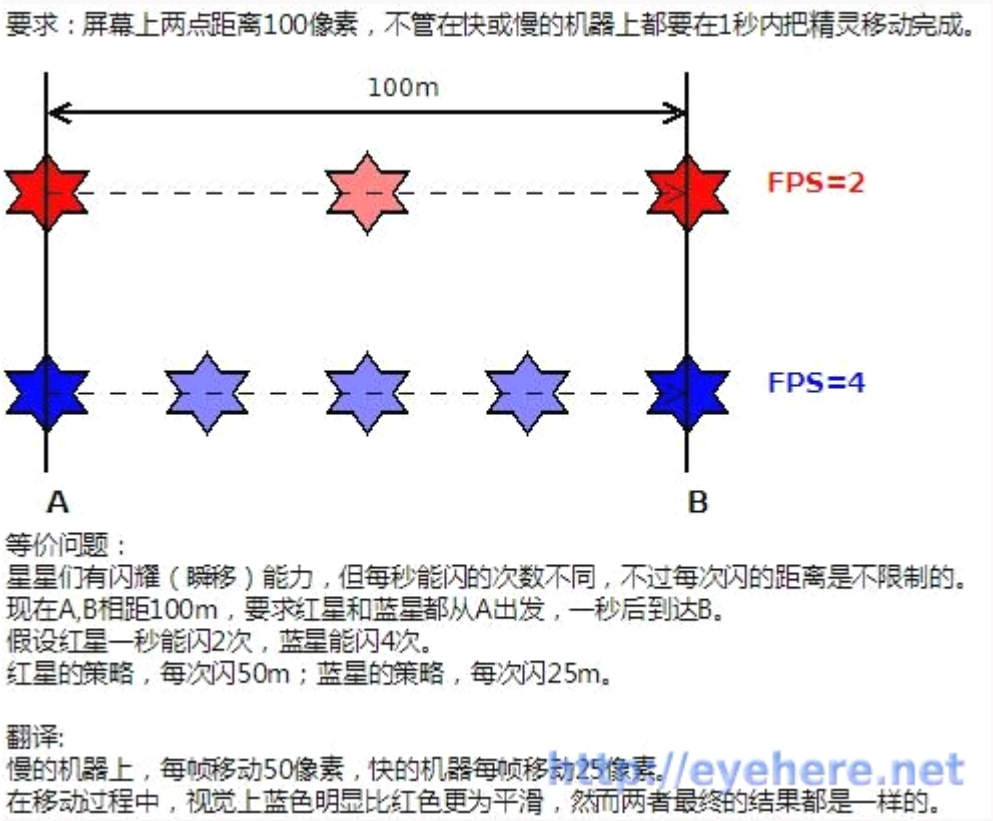
关于时间

有一个解决上述问题的方法，就是让我们的动画基于时间运作，我们需要知道上一个画面到现在经过了多少时间，然后我们才能决定是否开始绘制下一幅。pygame.time 模块给我们提供了一个 Clock 的对象，使我们可以轻易做到这一些：

```
clock=pygame.time.Clock()
time_passed=clock.tick()
time_passed=clock.tick(30)
```

第一行初始化了一个 Clock 对象；第二行的意识是返回一个上次调用的时间（以毫秒计）；第三行非常有用，在每一个循环中加上它，那么给 tick 方法加上的参数就成为了游戏绘制的最大帧率，这样的话，游戏就不会用掉你所有的 CPU 资源了！但是这仅仅是“最大帧率”，并不能代表用户看到的就是这个数字，有些时候机器性能不足，或者动画太复杂，实际的帧率达不到这个值，我们需要一种更有效的手段来控制我们的动画效果。

为了使得在不同机器上有着一致的效果，我们其实是需要给定物体（我们把这个物体叫做**精灵**，Sprite）恒定的速度。这样的话，从起点到终点的时间点是一样的，最终的效果也就相同了，所差别的，只是流畅度。看下面的图试着理解一下~



我们把上面的结论实际试用一下，假设让我们的小鱼儿每秒游动250像素，这样游动一个屏幕差不多需要2.56秒。我们就需要知道，从上一帧开始到现在，小鱼应该游动了多少像素，这个算法很简单，

速度*时间就行了，也就是250 * time_passed_second。不过我们刚刚得到的 time_passed 是毫秒，不要忘了除以1000.0，当然我们也能假设小鱼每毫秒游动0.25像素，这样就可以直接乘了，不过这样的速度单位有些怪怪的……

```
background_image_filename='sushiplate.jpg'
sprite_image_filename='fugu.png'
importpygame
frompygame.localsimport*
fromsysimportexit
pygame.init()
screen=pygame.display.set_mode((640,480),0,32)
background=pygame.image.load(background_image_filename).convert()
sprite=pygame.image.load(sprite_image_filename)
# Clock 对象
clock=pygame.time.Clock()
x=0.
# 速度（像素/秒）
speed=250.
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
exit()
screen.blit(background, (0,0))
screen.blit(sprite, (x,100))
time_passed=clock.tick()
time_passed_seconds=time_passed/1000.0
distance_moved=time_passed_seconds*speed
x+=distance_moved
# 想一下，这里减去640和直接归零有何不同？
ifx >640.:
x-=640.
pygame.display.update()
```

```
39
40
41
```

好了，这样不管你的机器是更深的蓝还是打开个记事本都要吼半天的淘汰机，人眼看起来，不同屏幕上的鱼的速度都是一致的了。请牢牢记住这个方法，在很多情况下，通过时间控制要比直接调节帧率好用的多。

斜线运动

下面有一个更有趣一些的程序，不再是单纯的直线运动，而是有点像屏保一样，碰到了壁会反弹。不过也并没有新的东西在里面，原理上来说，反弹只不过是把速度取反了而已~ 可以先试着自己写一个，然后与这个对照一下。

```
1  background_image_filename='sushiplate.jpg'
2  sprite_image_filename='fugu.png'
3  importpygame
4  frompygame.localsimport*
5  fromsysimportexit
6  pygame.init()
7  screen=pygame.display.set_mode((640,480),0,32)
8  background=pygame.image.load(background_image_filename).convert()
9  sprite=pygame.image.load(sprite_image_filename).convert_alpha()
10 clock=pygame.time.Clock()
11 x, y=100.,100.
12 speed_x, speed_y=133.,170.
13 whileTrue:
14 foreventinpygame.event.get():
15 ifevent.type==QUIT:
16 exit()
17 screen.blit(background, (0,0))
18 screen.blit(sprite, (x, y))
19 time_passed=clock.tick(30)
20 time_passed_seconds=time_passed/1000.0
21 x+=speed_x*time_passed_seconds
22 y+=speed_y*time_passed_seconds
23 # 到达边界则把速度反向
24 ifx >640-sprite.get_width():
25 speed_x=-speed_x
26 x=640-sprite.get_width()
27 elifx <0:
```

```
speed_x=-speed_x
x=0.
ify >480-sprite.get_height():
speed_y=-speed_y
y=480-sprite.get_height()
elify <0:
speed_y=-speed_y
y=0
pygame.display.update()
```

OK，这次的运动就说到这里。仔细一看的话，就会明白游戏中的所谓运动（尤其是2D游戏），不过是把一个物体的坐标改一下而已。不过总是不停的计算和修改x和y，有些麻烦不是么，下次我们引入向量，看看使用数学怎样可以帮我们减轻负担。

次我们说到了向量，不得不说向量是一个伟大的发明，在单纯的数字运算之中，居然就把方向也包含其中。对于如今的我们来看，非常普通的事情，几百年前的人们能够考虑到这个，实在是非常的不容易。不过同时我们也要有这样的意识——我们现在所使用的数学，未必就是最完美的。时代发展科技进步，或许我们会有更好的方式来诠释我们的世界。想想一片叶子飘落，有它独特的轨迹，如果要人类计算出来那个轨迹，即便可能，也是无比繁杂的。叶子懂我们的数学吗？不，它不懂，但它就优雅的落了下来。自然有着我们尚无法理解的思考方式，我们现在所使用的工具，还是太复杂！人类要向“道”继续努力才行啊。

“一方通行”凭借向量的力量成为了学园第一能力者，我们是否也应该好好学习向量？

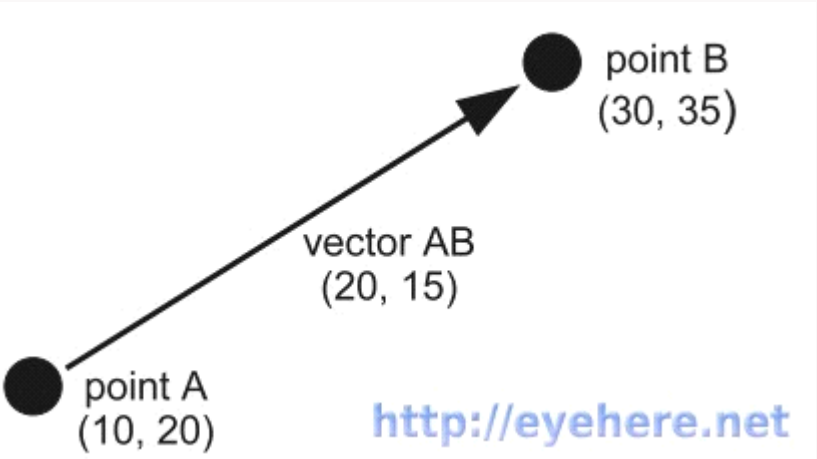
扯远了，虽然不记得学校里是什么时候开始接触到向量的，不过肯定也不会太晚，如果你不知道什么是向量，最好先找一本书看看吧，这里只会有一些最最核心的讲解。

引入向量

我们先考虑二维的向量，三维也差不多了，而游戏中的运动最多只用得到三维，更高的留给以后的游戏吧~

向量的表示和坐标很像，(10,20)对坐标而言，就是一个固定的点，然而在向量中，它意味着 x 方向行进10，y 方向行进20，所以坐标(0,0)加上向量(10,20)后，就到达了点(10,20)。

向量可以通过两个点来计算出来，如下图，A 经过向量 AB 到达了 B，则向量 AB 就是(30, 35) – (10, 20) = (20, 15)。我们也能猜到向量 BA 会是(-20, -15)，注意向量 AB 和向量 BA，虽然长度一样，但是方向不同。



在 Python 中，我们可以创建一个类来存储和获得向量（虽然向量的写法很像一个元组，但因为向量有很多种计算，必须使用类来完成）：

```
classVector2(object):
def__init__(self, x=0.0, y=0.0):
self.x=x
self.y=y
def__str__(self):
return"(%s, %s)"%(self.x,self.y)
@classmethod
deffrom_points(cls, P1, P2):
returncls( P2[0] – P1[0], P2[1] – P1[1] )
#我们可以使用下面的方法来计算两个点之间的向量
A=(10.0,20.0)
B=(30.0,35.0)
AB=Vector2.from_points(A, B)
printAB

```

原理上很简单，函数修饰符@不用我说明了吧？如果不明白的话，可以参考 Python 的编程指南。

向量的大小

向量的大小可以简单的理解为那根箭头的长度，勾股定理熟稔的各位立刻知道怎么计算了：

```
defget_magnitude(self):
returnmath.sqrt(self.x**2+self.y**2)

```

把这几句加入到刚刚的 Vector2里，我们的向量类就多了计算长度的能力。嗯，别忘了一开始要引入 math 库。

单位向量

一开头说过，向量有着大小和方向两个要素，通过刚刚的例子，我们可以理解这两个意思了。在向量的大家族里，有一种比较特殊的向量叫“**单位向量**”，意思是大小为1的向量，我们还能把任意向量方向不变的缩放（体现在数字上就是x和y等比例的缩放）到一个单位向量，这叫向量的**规格（正规）化**，代码体现的话：

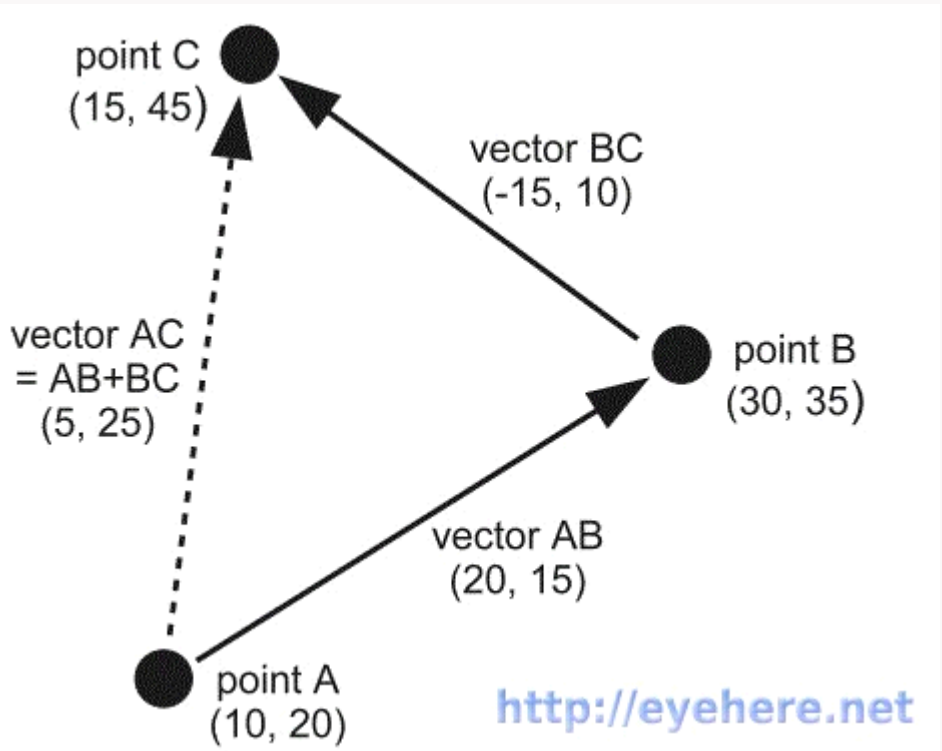
```
defnormalize(self):
magnitude=self.get_magnitude()
self.x/=magnitude
self.y/=magnitude

```

使用过 normalize 方法以后，向量就成了一个单位向量。单位向量有什么用？我们以后会看到。

向量运算

我们观察下图，点 B 由 A 出发，通过向量 AB 到达，C 则有 B 到达，通过 BC 到达；C 直接由 A 出发的话，就得经由向量 AC。



由此我们得到一个显而易见的结论向量 AC = 向量 AB + 向量 BC。向量的加法计算方法呼之欲出：

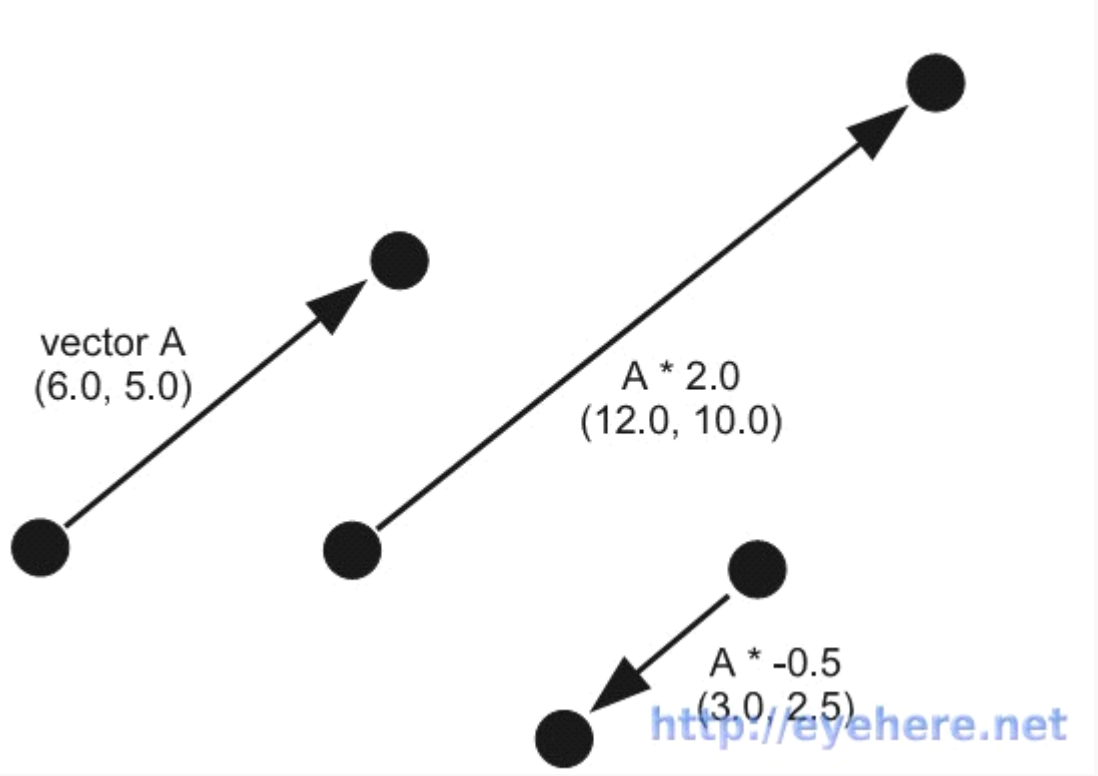
(20, 15) + (-15, 10) = (20-15, 15+10) = (5, 25)

把各个方向分别相加，我们就得到了向量的加法运算法则。很类似的，减法也是同样，把各个方向分别想减，可以自己简单验证一下。代码表示的话：

```
def__add__(self, rhs):
returnVector2(self.x+rhs.x,self.y+rhs.y)
def__sub__(self, rhs):
returnVector2(self.x-rhs.x,self.y-rhs.y)
```

两个下划线“__”为首尾的函数，在 Python 中一般就是**重载**的意思，如果不知道的话还需要稍微努力努力：）当然，功力稍深厚一点的，就会知道这里 super 来代替 Vector2可能会更好一些，确实如此。不过这里只是示例代码，讲述一下原理而已。

有加减法，那乘除法呢？当然有！不过向量的乘除并不是发生在两个向量直接，而是用一个向量来乘/除一个数，其实际意义就是，向量的方向不变，而大小放大/缩小多少倍。如下图：




```
def__mul__(self, scalar):
returnVector2(self.x*scalar,self.y*scalar)
def__div__(self, scalar):
returnVector2(self.x/scalar,self.y/scalar)
```

向量的运算被广泛的用来计算到达某个位置时的中间状态，比如我们知道一辆坦克从 A 到 B，中间有10帧，那么很显然的，把步进向量通过(B-A)/10计算出来，每次在当前位置加上就可以了。很简单吧？

更好的向量类

我们创造的向量类已经不错了，不过毕竟只能做一些简单的运算，别人帮我们已经写好了更帅的库（早点不拿出来？写了半天…… 原理始终是我们掌握的，自己动手，印象更深），是发挥拿来主义的时候了（可以尝试使用 easy_install gameobjects 简单的安装起来）。下面是一个使用的例子：

```
1  fromgameobjects.vector2import*
2  A=(10.0,20.0)
3  B=(30.0,35.0)
4  AB=Vector2.from_points(A, B)
5  print"Vector AB is", AB
6  print"AB * 2 is", AB*2
7  print"AB / 2 is", AB/2
8  print"AB + (–10, 5) is", AB+(–10,5)
9  print"Magnitude of AB is", AB.get_magnitude()
10 print"AB normalized is", AB.get_normalized()
11 # 结果是下面
12 Vector ABis(20,15)
13 AB*2is(40,30)
14 AB/2is(10,7.5)
```

```
15  AB+(-10,5)is(10,20)

16  Magnitude of ABis25.0

17  AB normalizedis(0.8,0.6)

18
```

使用向量的游戏动画

终于可以实干一番了！这个例子比我们以前写的都要帅的多，小鱼不停的在我们的鼠标周围游动，若即若离：

```
1
2
3
4
5
6   background_image_filename='sushiplate.jpg'
7   sprite_image_filename='fugu.png'
8   importpygame
9   frompygame.localsimport*
10  fromsysimportexit
11  fromgameobjects.vector2importVector2
12  pygame.init()
13  screen=pygame.display.set_mode((640,480),0,32)
14  background=pygame.image.load(background_image_filename).convert()
15  sprite=pygame.image.load(sprite_image_filename).convert_alpha()
16  clock=pygame.time.Clock()
17  position=Vector2(100.0,100.0)
18  heading=Vector2()
19  whileTrue:
20  foreventinpygame.event.get():
21  ifevent.type==QUIT:
22  exit()
23  screen.blit(background, (0,0))
24  screen.blit(sprite, position)
25  time_passed=clock.tick()
26  time_passed_seconds=time_passed/1000.0
27  # 参数前面加*意味着把列表或元组展开
28  destination=Vector2(*pygame.mouse.get_pos() )-Vector2(*sprite.get_
29  size() )/2
30  # 计算鱼儿当前位置到鼠标位置的向量
31  vector_to_mouse=Vector2.from_points(position, destination)
32  # 向量规格化
33  vector_to_mouse.normalize()
34  # 这个 heading 可以看做是鱼的速度，但是由于这样的运算，鱼的速度就不断改变
35  了
36  # 在没有到达鼠标时，加速运动，超过以后则减速。因而鱼会在鼠标附近晃动。
37  heading=heading+(vector_to_mouse*.6)
38  position+=heading*time_passed_seconds
39  pygame.display.update()
40
41
42
43
44
```

虽然这个例子里的计算有些让人看不明白，但是很明显 heading 的计算是关键，如此复杂的运动，使用向量居然两句话就搞定了~看来没有白学。

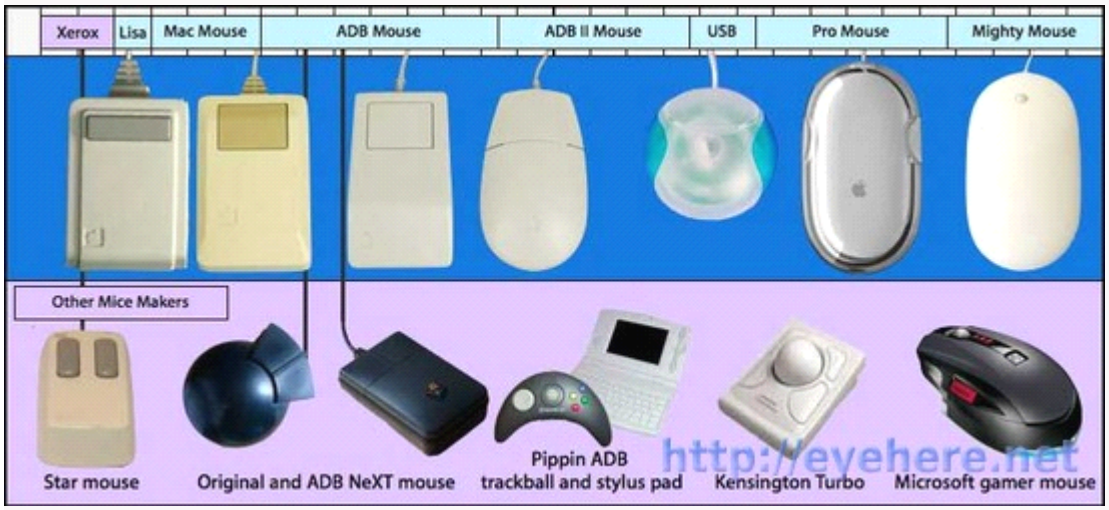
动画总结

- 正如上一章所说，所谓动画，不过是在每一帧上，相对前一帧把精灵的坐标在加减一些而已；
- 使用时间来计算加减的量以在不同性能的计算机上获得一致的动画效果；
- 使用向量来计算运动的过程来减轻我们的劳动，在3D 的情况下，简单的使用 Vector3 便可以了。

如今我们已经学习到了游戏动画制作的精髓，一旦可以动起来，就能创造无数让人叹为观止的效果，是不是应该写个程序在朋友们面前炫耀炫耀了？
在下面，我们要学习接受输入和游戏里的物体互动起来。

有时候无聊在网上翻翻小说看看，绝大多数那叫一个无聊。比如说修炼的境界分几种，都有个名字，然后每种境界再有几层，这不就是变相的打怪练级么？文笔也不咋样，故事情节的驾驭能力更是让我瞠目结舌，想到这些类小说盛行，不觉感到悲从中来。感觉看这些小说，就想在看别人打游戏一般，崩溃到极点。游戏和小说的最大区别，除了声色以外，最不同的就是玩家可以沉入进去，通过自己的双手来参与；而好的游戏，更是可以通过玩家的选择，完全掌控游戏的发展，这是传统的故事媒介无法做到的事情。

上次我们说明了使用键盘操作游戏，键盘是非常古老的输入设备，甚至笔计算机本身都要古老的多，因为它发源于打字机，貌似1868年就有成熟的打字机问世了。不得不说的是，现在最常用的键位排部，并不是最科学的，相比上一次说过的 DUORAK 键盘，打字者的手指平均每日运动1英里，而 QWERTY 则是12到20英里。当然这对游戏毫无意义……



相比而言，鼠标非常的年轻，世界上最早的鼠标诞生于1964年，它是由美国人道格·恩格尔巴特（Doug Engelbart）发明的。IEEE协会把鼠标的发明列为计算机诞生50年来最重大的事件之一，可见其对IT历程的重大影响作用。1983年苹果公司给自家的电脑安上了鼠标，用户就开始离不开这个小东西了。而现代游戏，离开了鼠标，99%的都没法玩！我们自然得好好研究如何使用鼠标来操控我们的游戏。

使用鼠标控制精灵

我们已经看到如何画一个光标了，只是简单的在鼠标坐标上画一个图像而已，我们可以从MOUSEMOTION或者pygame.mouse.get_pos方法来获得坐标。但我们还可以使用这个坐标来控制方向，比如在3D游戏中，可以使用鼠标来控制视角。这种时候，我们不使用鼠标的位置，因为鼠标可能会跑到窗口外面，我们使用鼠标现在与上一帧的相对偏移量。在下一个例子中，我们演示使用鼠标的左右移动来转动我们熟悉的小鱼儿：

```
background_image_filename='sushiplate.jpg'
sprite_image_filename='fugu.png'
importpygame
frompygame.localsimport*
fromsysimportexit
fromgameobjects.vector2importVector2
frommathimport*
pygame.init()
screen=pygame.display.set_mode((640,480),0,32)
background=pygame.image.load(background_image_filename).convert()
sprite=pygame.image.load(sprite_image_filename).convert_alpha()
clock=pygame.time.Clock()
# 让 pygame 完全控制鼠标
pygame.mouse.set_visible(False)
pygame.event.set_grab(True)
sprite_pos=Vector2(200,150)
sprite_speed=300.
sprite_rotation=0.
sprite_rotation_speed=360.
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
```

```
exit()
# 按 Esc 则退出游戏
ifevent.type==KEYDOWN:
ifevent.key==K_ESCAPE:
exit()
pressed_keys=pygame.key.get_pressed()
# 这里获取鼠标的按键情况
pressed_mouse=pygame.mouse.get_pressed()
rotation_direction=0.
movement_direction=0.
# 通过移动偏移量计算转动
rotation_direction=pygame.mouse.get_rel()[0]/5.0
ifpressed_keys[K_LEFT]:
rotation_direction=+1.
ifpressed_keys[K_RIGHT]:
rotation_direction=-1.
# 多了一个鼠标左键按下的判断
ifpressed_keys[K_UP]orpressed_mouse[0]:
movement_direction=+1.
# 多了一个鼠标右键按下的判断
ifpressed_keys[K_DOWN]orpressed_mouse[2]:
movement_direction=-1.
screen.blit(background, (0,0))
rotated_sprite=pygame.transform.rotate(sprite, sprite_rotation)
w, h=rotated_sprite.get_size()
sprite_draw_pos=Vector2(sprite_pos.x-w/2, sprite_pos.y-h/2)
screen.blit(rotated_sprite, sprite_draw_pos)
time_passed=clock.tick()
time_passed_seconds=time_passed/1000.0
sprite_rotation+=rotation_direction*sprite_rotation_speed*time_passed_seconds
heading_x=sin(sprite_rotation*pi/180.)
heading_y=cos(sprite_rotation*pi/180.)
heading=Vector2(heading_x, heading_y)
heading*=movement_direction
sprite_pos+=heading*sprite_speed*time_passed_seconds
pygame.display.update()
```

```
67
68
69
70
71
72
73
74
75
76
77
```

一旦打开这个例子，鼠标就看不到了，我们得使用 Esc 键来退出程序，除了上一次的方向键，当鼠标左右移动的时候，小鱼转动，按下鼠标左右键的时候，小鱼前进/后退。看代码，基本也是一样的，就多了几句带注释的。

这里使用了

```
pygame.mouse.set_visible(False)
pygame.event.set_grab(True)
```

来完全控制鼠标，这样鼠标的光标看不见，也不会跑到 pygame 窗口外面去，一个副作用就是无法使用鼠标关闭窗口了，所以你得准备一句代码来退出程序。

然后我们使用

```
rotation_direction = pygame.mouse.get_rel()[0] / 5.
```

来获得 x 方向上的偏移量，除以5是把动作放慢一点……

还有

```
lmb, mmb, rmb = pygame.mouse.get_pressed()
```

获得了鼠标按键的情况，如果有一个按键按下，那么对应的值就会为 True。

总结一下 pygame.mouse 的函数：

- pygame.mouse.get_pressed —— 返回按键按下情况，返回的是一元组，分别为(左键，中键，右键)，如按下则为 True
- pygame.mouse.get_rel —— 返回相对偏移量，(x 方向, y 方向)的一元组
- pygame.mouse.get_pos —— 返回当前鼠标位置(x, y)

- pygame.mouse.set_pos —— 显而易见，设置鼠标位置
- pygame.mouse.set_visible —— 设置鼠标光标是否可见
- pygame.mouse.get_focused —— 如果鼠标在 pygame 窗口内有效，返回 True
- pygame.mouse.set_cursor —— 设置鼠标的默认光标式样 ,是不是感觉我们以前做的事情白费了？哦不会，我们使用的方法有着更好的效果。
- pyGame.mouse.get_cursor —— 不再解释。

关于使用鼠标

在游戏中活用鼠标是一门学问，像在 FPS 中，鼠标用来瞄准，ARPG 或 RTS 中，鼠标用来指定位置和目标。而在很多策略型的小游戏中，鼠标的威力更是被发挥的 淋漓尽致，也许是可以放置一些道具，也许是用来操控蓄力。我们现在使用的屏幕是二维的，而鼠标也能在2维方向到达任何的位置，所以鼠标相对键盘，更适合现代的复杂操作，只有想不到没有做不到啊。

绝大多数时候，鼠标和键盘是合作使用的，比如使用键盘转换视角，使用键盘移动，或者键盘对应很多快捷键，而键盘则用来指定位置。开动大脑，创造未来！

很自然的，我们讲述了游戏中视觉上的种种，现在开始就要学习一下游戏中的用户输入。同样我们也要探讨一下如何让用户的输入更为顺畅，换个词就是，如果让游戏的手感更好一些。

游戏设备

玩过游戏的都知道鼠标和键盘是游戏的不可或缺的输入设备。键盘可以控制有限的方向和诸多的命令操作，而鼠标更是提供了全方位的方向和位置操作。不过这两个设备并不是为游戏而生，专业的游戏手柄给玩家提供了更好的操作感，加上力反馈等技术，应该说游戏设备越来越丰富，玩家们也是越来越幸福。

键盘设备

我们先从最广泛的键盘开始讲起。

现在使用的键盘，基本都是 QWERTY 键盘（看看字幕键盘排布的左上就知道了），尽管这个世界上还有其他种类的键盘，比如 AZERTY 啥的，反正我是没见过，如果你能在写游戏的时候考虑到这些特殊用户自然是最好，个人感觉是问题不大吧。

以前第二部分也稍微使用了一下键盘，那时候是用了 **pygame.event.get()**获取所有的事件，当 **event.type == KEYDOWN** 的时候，在判断 **event.key** 的种类，而各个种类也使用 **K_a**，**K_b**……等判断。这里再介绍一个 **pygame.key.get_pressed()**来获得所有按下的键值，它会返回一个元组，这个元组的索引就是键值，对应的就是是否按下，比如说：

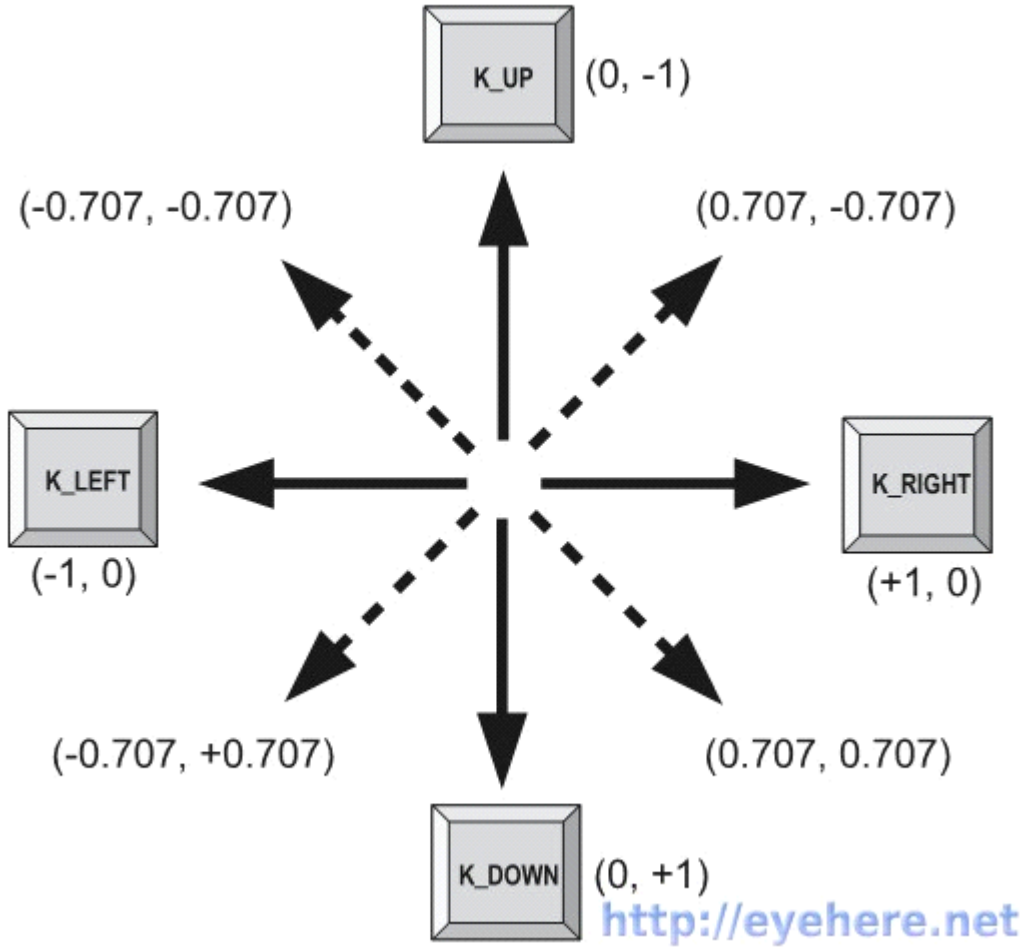
```
pressed_keys=pygame.key.get_pressed()
ifpressed_keys[K_SPACE]:
# Space key has been pressed
fire()pressed_keys=pygame.key.get_pressed()
```

当然 key 模块下还有很多函数：

- key.get_focused —— 当前激活的 pygame 窗口
- key.get_pressed —— 刚刚解释过了
- key.get_mods —— 按下的组合键（Alt, Ctrl, Shift）
- key.set_mods —— 你也可以模拟按下组合键的效果（KMOD_ALT, KMOD_CTRL, KMOD_SHIFT）
- key.set_repeat —— 设定允许 pygame 接受重复按键
- key.name —— 接受键值返回键名

使用键盘控制方向

有了上一章向量的基础，只需一幅图就能明白键盘如何控制方向：



很多游戏也使用 ASDW 当做方向键来移动，我们来看一个实际的例子：

```
background_image_filename='sushiplate.jpg'
sprite_image_filename='fugu.png'
importpygame
frompygame.localsimport*
fromsysimportexit
fromgameobjects.vector2importVector2
pygame.init()
screen=pygame.display.set_mode((640,480),0,32)
background=pygame.image.load(background_image_filename).convert()
sprite=pygame.image.load(sprite_image_filename).convert_alpha()
clock=pygame.time.Clock()
sprite_pos=Vector2(200,150)
sprite_speed=300.
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
exit()
```

```
pressed_keys=pygame.key.get_pressed()
key_direction=Vector2(0,0)
ifpressed_keys[K_LEFT]:
key_direction.x=-1
elifpressed_keys[K_RIGHT]:
key_direction.x=+1
ifpressed_keys[K_UP]:
key_direction.y=-1
elifpressed_keys[K_DOWN]:
key_direction.y=+1
key_direction.normalize()
screen.blit(background, (0,0))
screen.blit(sprite, sprite_pos)
time_passed=clock.tick(30)
time_passed_seconds=time_passed/1000.0
sprite_pos+=key_direction*sprite_speed*time_passed_seconds
pygame.display.update()
```

这个例子很简单，就是使用方向键移动小鱼。使用的知识也都讲过了，相信大家都可以理解。不过这里并不是单纯的判断按下的键来获得方向，而是通过对方向的加减来获得最终的效果，这样可能会更简短一些，也需要一些技术；如果把方向写入代码，效率更高，不过明显通用性就要低一些。记得把力气花在刀刃上！当然这个例子也不是那么完美，看代码、实践一下都能看到，左方向键的优先级大于右方向键，而上则优于下，我们是否有更好的方法？……有兴趣的自己考虑~

这个例子我们可以看到，小鱼只能在八个方向移动，如何做到全方向？如果你游戏经验足一点或许可

以想到，是的，先转向，再移动，尽管不是那么快捷，但毕竟达到了目标。我们看一下这样的代码怎么写：

```
background_image_filename='sushiplate.jpg'
sprite_image_filename='fugu.png'
importpygame
frompygame.localsimport*
fromsysimportexit
fromgameobjects.vector2importVector2
frommathimport*
pygame.init()
screen=pygame.display.set_mode((640,480),0,32)
background=pygame.image.load(background_image_filename).convert()
sprite=pygame.image.load(sprite_image_filename).convert_alpha()
clock=pygame.time.Clock()
sprite_pos=Vector2(200,150)# 初始位置
sprite_speed=300.# 每秒前进的像素数（速度）
sprite_rotation=0.# 初始角度
sprite_rotation_speed=360.# 每秒转动的角度数（转速）
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
exit()
pressed_keys=pygame.key.get_pressed()
rotation_direction=0.
movement_direction=0.
# 更改角度
ifpressed_keys[K_LEFT]:
rotation_direction=+1.
ifpressed_keys[K_RIGHT]:
rotation_direction=-1.
# 前进、后退
ifpressed_keys[K_UP]:
movement_direction=+1.
ifpressed_keys[K_DOWN]:
movement_direction=-1.
screen.blit(background, (0,0))
# 获得一条转向后的鱼
rotated_sprite=pygame.transform.rotate(sprite, sprite_rotation)
# 转向后，图片的长宽会变化，因为图片永远是矩形，为了放得下一个转向后的矩形，外接的矩形势必会比较大
w, h=rotated_sprite.get_size()
# 获得绘制图片的左上角（感谢 pltc325网友的指正）
```

```
sprite_draw_pos=Vector2(sprite_pos.x-w/2, sprite_pos.y-h/2)
screen.blit(rotated_sprite, sprite_draw_pos)
time_passed=clock.tick()
time_passed_seconds=time_passed/1000.0
# 图片的转向速度也需要和行进速度一样，通过时间来控制
sprite_rotation+=rotation_direction*sprite_rotation_speed*time_passed_seconds
# 获得前进（x 方向和 y 方向），这两个需要一点点三角的知识
heading_x=sin(sprite_rotation*pi/180.)
heading_y=cos(sprite_rotation*pi/180.)
# 转换为单位速度向量
heading=Vector2(heading_x, heading_y)
# 转换为速度
heading*=movement_direction
sprite_pos+=heading*sprite_speed*time_passed_seconds
pygame.display.update()
```

我们通过上下控制前进/后退，而左右控制转向。我们通过 **pygame.transform.rotate()**来获得了转向后的图片，具体参数可以参考代码。各条语句的作用也可以参考注释。

下次讲解使用鼠标控制游戏。

用 Python 和 Pygame 写游戏-从入门到精通（13）

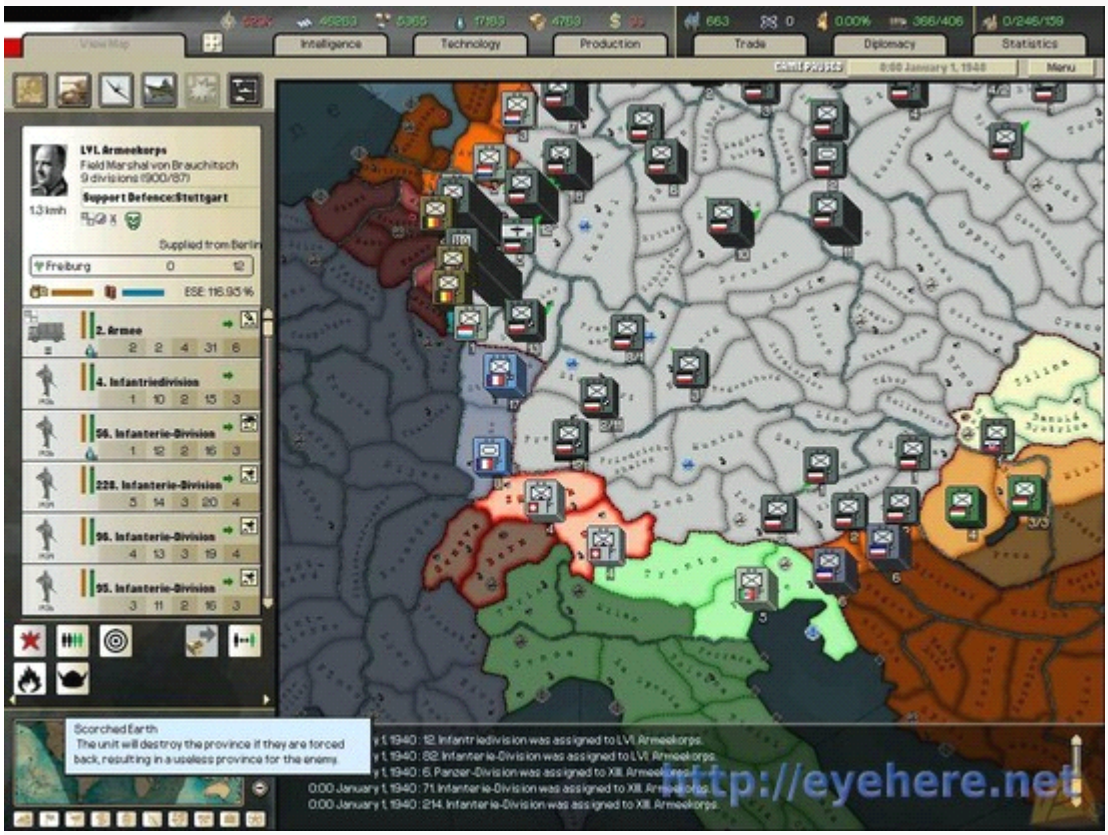
星期六, 9. 七月 2011

我们已经学习了游戏的图像和输入编程，接下来是什么？声音？没错，不过我们要晚一点再说声音。

游戏中还有什么那么重要？哦是的，趣味性。

游戏的趣味是游戏生命的重要组成部分，其重要性甚至凌驾于任何元素，包括画面之上，之所以大家不太想得到，是因为这是个比较难定量的东西，难以直接用个什么方法测量出来。那么支持游戏趣味的是什么呢？是**规则**和**智能**。

规则是游戏的玩法，比如围棋，尽量扩大自己的领地；比如俄罗斯方块，把不同的方块组合起来，不留缝隙；再比如粘土世界，把小球连起来到达目的地。这就是游戏规则的具体体现，好的规则简单让人很容易理解，但是又能产生足够多的变化让我们无法穷尽它的方方面面。书写好的规则是天才或者干脆上天干的事，本系列文章还无法触及，请读者自己修炼了……



我们要学习游戏的另外一个支撑物，智能，或者帅气一点称为 **AI**（Artificial Intelligence，人工智能，因为游戏里的智能肯定是人赋予的）。玩家操作我们自己的角色，那么 NPC（nonplayer characters）呢？交由 AI 去操作，所以如果游戏中有何你相同地位的角色存在的话，你就是在和 AI 对垒。智能意味着对抗，“与人斗其乐无穷”，就是因为人足够聪明，要想“玩游戏其乐无穷”，我们都得赋予游戏足够的 AI。

为游戏创建人工智能

也许你希望能在 Pygame 中发现一个 pygame.ai 模块，不过每个游戏中的智能都是不同的，很难准备一个通用的模块。一个简单的游戏中并不需要多少 AI 编程的代码，比如俄罗斯方块，你只需要随机的落下一个方块组合，然后每次下降完毕扫描一下落下的方块就好了，这甚至不能称为 AI。但比如魔兽争霸，这里面的 AI 就非常的复杂，一般人都要学习一段时间才能打败电脑，可想而知其高度了。

尽管一般游戏中的人工智能都是用来对付人类的，不过随着游戏发展，AI 也可能是我们朋友，甚至 AI 互相影响从而改变整个游戏世界，这样的游戏就有了更多的深度和未知，无法预知的东西总是吸引人的不是么？

游戏的 AI 编程不是一件简单的事情，幸运的是 AI 代码往往可以重用，这个我们以后再讲。

我们接下来要讲述游戏 AI 的技术，赋予游戏角色以生命，应该说人工智能是很高端的技术，花费几十年都未必能到达怎么的一个高度，所以这里的几章还是以讲解重要概念为主。作为参考，个人推荐 Mat Buckland 的《AI Techniques for Game Programming》，中文版《游戏编程中的人工智能技术》由清华大学出版社出版，很不错的一本入门书籍。

什么是人工智能

出于严谨性，我们应该给人工智能下一个定义。每个人都会对智能有一个概念，但又很难给它下一个确切的定义。著名的美国斯坦福大学人工智能研究中心尼尔逊教授对人工智能下了这样一个定义：“人工智能是关于知识的学科——怎样表示知识以及怎样获得知识并使用知识的科学。”而另一个美国麻省理工学院的温斯顿教授认为：“人工智能就是研究如何使计算机去做过去只有人才能做的智能工作。”这些说法反映了人工智能学科的基本思想和基本内容。即人工智能是研究人类智能活动的规律，构造具有一定智能的人工系统，研究如何让计算机去完成以往需要人的智力才能胜任的工作，也就是研究

如何应用计算机的软硬件来模拟人类某些智能行为的基本理论、方法和技术。但这些说辞太麻烦了，我觉得，人工智能就是自我感知和反应的人造系统，足矣。

智能是一件玄妙的事情，在游戏中的人工智能更是如此，我们用程序中的一些数据结构和算法就构筑了 NPC 的大脑，听起来太酷了！更酷的是，Python 非常适合用来编写人工智能。

人工智能初探

举超级玛丽为一个例子，那些走来走去的老乌龟，我们控制英雄踩到它们头上就能杀死它们，而平时，它们就在两根管子之间走来走去（这样的人生真可悲……），如果我们打开它们的脑袋看一下，可能会看到这样的代码：

```
self.move_forward()
ifself.hit_wall():
self.change_direction()
```

无比简单，向前走，一撞墙就回头，然后重复。它只能理解一种状态，就是**撞墙**，而一旦到达这个状态，它的反应就是回头。

在考虑一个会发子弹的小妖怪，它的脑袋可能是这么长的：

```
ifself.state=="exploring":
self.random_heading()
ifself.can_see(player):
self.state="seeking"
elifself.state=="seeking":
self.head_towards("player")
ifself.in_range_of(player):
self.fire_at(player)
ifnotself.can_see(player):
self.state="exploring"
```

它就有了两个状态，**搜寻**和**锁定**。如果正在搜寻，就随处走动，如果发现目标，就**锁定**他，然后靠近并试着向其开火，而一旦丢失目标（目标走出了视线范围或者被消灭），重新进入搜寻状态。这个 AI 也是很简单的，但是它对玩家来说就有了一定的危险性。如果我们给它加入更多的状态，它就会变得

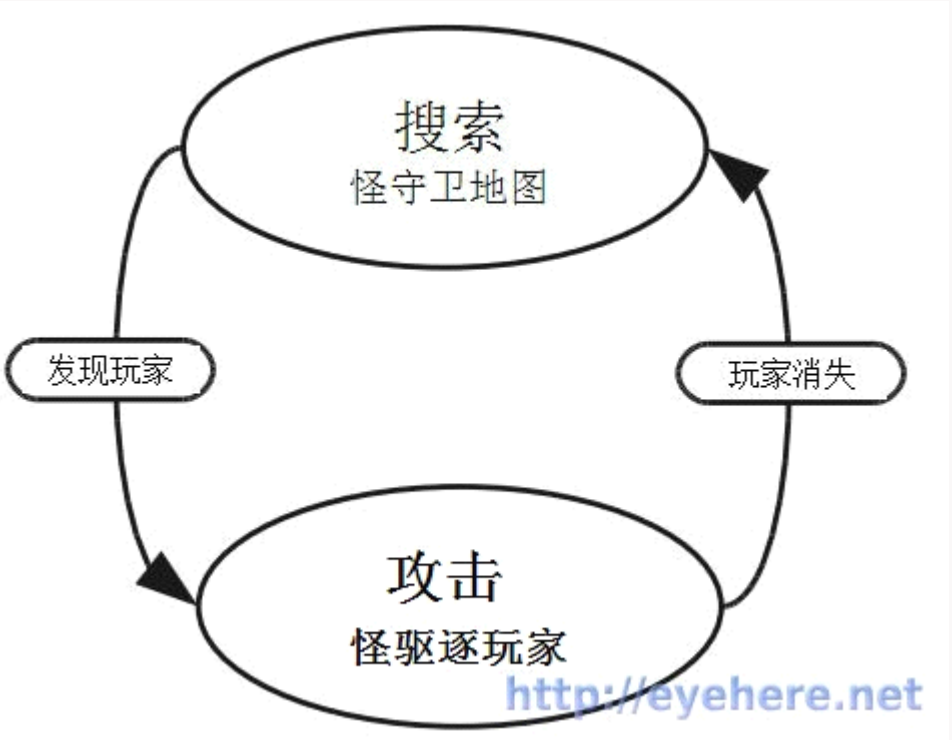
更厉害，游戏的趣味性也就可能直线上扬了。

OK，这就是我们下一次要讲的主题，**状态机**。

用 Python 和 Pygame 写游戏-从入门到精通（14）

星期四, 14. 七月 2011

上一次稍微说了一下 AI，为了更好的理解它，我们必须明白什么是状态机。有限状态机（英语：finite-state machine, FSM），又称有限状态自动机，简称状态机，是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。太抽象了，我们看看上一次的机器人的状态图，大概是长的这个样子：



状态定义了两个内容：

- 当前正在做什么
- 转化到下一件事时候的条件

状态同时还可能包含**进入（entry）**和**退出(exit)**两种动作，进入时间是指进入某个状态时要做的一次性的事情，比如上面的怪，一旦进入攻击状态，就得开始计算与玩家的距离，或许还得大吼一声“我要杀了你”等等；而退出动作则是与之相反的，离开这个状态要做的事情。

我们来创建一个更为复杂的场景来阐述这个概念——一个蚁巢世界。我们常常使用昆虫来研究 AI，因为昆虫的行为很简单容易建模。在我们这次的环境里，有三个实体(entity)登场：叶子、蜘蛛、蚂蚁。叶子会随机的出现在屏幕的任意地方，并由蚂蚁回收至蚁穴，而蜘蛛在屏幕上随便爬，平时蚂蚁不会在意它，而一旦进入蚁穴，就会遭到蚂蚁的极力驱赶，直至蜘蛛挂了或远离蚁穴。

尽管我们是对昆虫建模的，这段代码对很多场景都是合适的。把它们替换为巨大的机器人守卫（蜘蛛），坦克（蚂蚁）、能源（叶子），这段代码依然能够很好的工作。

游戏实体类

这里出现了三个实体，我们试着写一个通用的实体基类，免得写三遍了，同时如果加入了其他实体，也能很方便的扩展出来。

一个实体需要存储它的名字，现在的位置，目标，速度，以及一个图形。有些实体可能只有一部分属性（比如叶子不应该在地图上瞎走，我们把它的速度设为0），同时我们还需要准备进入和退出的函数供调用。下面是一个完整的 GameEntity 类：

```
classGameEntity(object):
def__init__(self, world, name, image):
self.world=world
self.name=name
self.image=image
self.location=Vector2(0,0)
self.destination=Vector2(0,0)
self.speed=0.
self.brain=StateMachine()
self.id=0
defrender(self, surface):
x, y=self.location
w, h=self.image.get_size()
surface.blit(self.image, (x-w/2, y-h/2))
defprocess(self, time_passed):
self.brain.think()
ifself.speed >0andself.location !=self.destination:
vec_to_destination=self.destination-self.location
```

```
distance_to_destination=vec_to_destination.get_length()
heading=vec_to_destination.get_normalized()
travel_distance=min(distance_to_destination, time_passed*self.speed)
self.location+=travel_distance*heading
```

观察这个类，会发现它还保存一个 world，这是对外界描述的一个类的引用，否则实体无法知道外界的信息。这里类还有一个 id，用来标示自己，甚至还有一个 brain，就是我们后面会定义的一个状态机类。

render 函数是用来绘制自己的。

process 函数首先调用 self.brain.think 这个状态机的方法来做一些事情（比如转身等），。接下来的代码用来让实体走近目标。

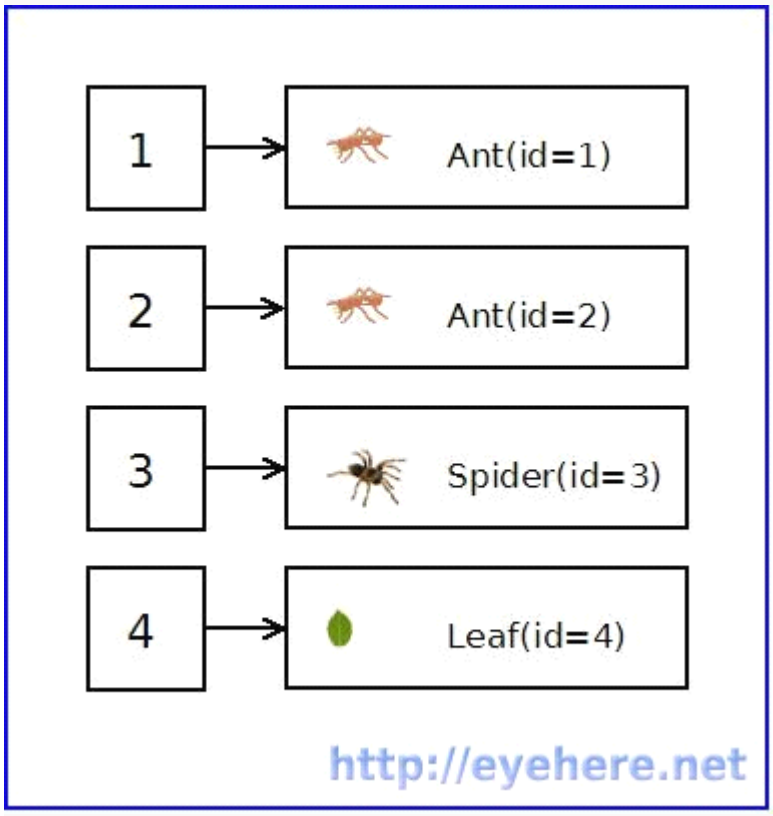
世界类

我们写了一个 GameObject 的实体类，这里再有一个世界类 World 用来描述外界。这里的世界不需要多复杂，仅仅需要准备一个蚁穴，和存储若干的实体位置就足够了：

```
classWorld(object):
def__init__(self):
self.entities={}# Store all the entities
self.entity_id=0# Last entity id assigned
# 画一个圈作为蚁穴
self.background=pygame.surface.Surface(SCREEN_SIZE).convert()
self.background.fill((255,255,255))
pygame.draw.circle(self.background, (200,255,200), NEST_POSITION,int(NEST_SIZE))
defadd_entity(self, entity):
# 增加一个新的实体
self.entities[self.entity_id]=entity
entity.id=self.entity_id
self.entity_id+=1
defremove_entity(self, entity):
delself.entities[entity.id]
defget(self, entity_id):
# 通过 id 给出实体，没有的话返回 None
ifentity_idinself.entities:
returnself.entities[entity_id]
else:
returnNone
```

```
defprocess(self, time_passed):
# 处理世界中的每一个实体
time_passed_seconds=time_passed/1000.0
forentityinself.entities.itervalues():
entity.process(time_passed_seconds)
defrender(self, surface):
# 绘制背景和每一个实体
surface.blit(self.background, (0,0))
forentityinself.entities.values():
entity.render(surface)
defget_close_entity(self, name, location,range=100.):
# 通过一个范围寻找之内的所有实体
location=Vector2(*location)
forentityinself.entities.values():
ifentity.name==name:
distance=location.get_distance_to(entity.location)
ifdistance <range:
returnentity
returnNone
```

因为我们有着一系列的 GameObject，使用一个列表来存储就是很自然的事情。不过如果实体增加，搜索列表就会变得缓慢 所以我们使用了字典来存储。我们就使用 GameObject的 id 作为字典的key，实例作为内容来存放，实际的样子会是这样：



大多数的方法都用来管理实体，比如 add_entity 和 remove_entity。process 方法是用来调用所有试题的 process，让它们更新自己的状态；而 render 则用来绘制这个世界；最后 get_close_entity 用来寻找某个范围内的实体，这个方法会在实际模拟中用到。

这两个类还不足以构筑我们的昆虫世界，但是却是整个模拟的基础，下一次我们就要讲述实际的蚂蚁类和大脑（状态机类）。

用 Python 和 Pygame 写游戏-从入门到精通（15）

星期一, 18. 七月 2011

在继续我们的 AI 之旅前，分享一个在煎蛋上看到的有趣新闻，能通过读说明书来学习的 AI，这个世界真是变得越来越不可琢磨啦！机器人很快就要超越咱了……

因为这一次是接着上面的内容的，所以请不要跳过直接看这里。

哭！！！写完了上传出错，丢失啊，重伤重写~~~~~~~~~~~~

蚂蚁实例类

在我们正式建造大脑之前，我们得先做一个蚂蚁类出来，就是下面的这个，从 GameEntity 继承而来：

```
classAnt(GameEntity):
def__init__(self, world, image):
# 执行基类构造方法
GameEntity.__init__(self, world,"ant", image)
# 创建各种状态
exploring_state=AntStateExploring(self)
seeking_state=AntStateSeeking(self)
delivering_state=AntStateDelivering(self)
hunting_state=AntStateHunting(self)
self.brain.add_state(exploring_state)
self.brain.add_state(seeking_state)
self.brain.add_state(delivering_state)
```

```
13 self.brain.add_state(hunting_state)
14 self.carry_image=None
15 defcarry(self, image):
16 self.carry_image=image
17 defdrop(self, surface):
18 # 放下 carry 图像
19 ifself.carry_image:
20 x, y=self.location
21 w, h=self.carry_image.get_size()
22 surface.blit(self.carry_image, (x-w, y-h/2))
23 self.carry_image=None
24 defrender(self, surface):
25 # 先调用基类的 render 方法
26 GameEntity.render(self, surface)
27 # 额外绘制 carry_image
28 ifself.carry_image:
29 x, y=self.location
30 w, h=self.carry_image.get_size()
31 surface.blit(self.carry_image, (x-w, y-h/2))
```

这个 Ant 类先调用了父类的__init__，都是 Python 基础不多说了。下面的代码就是一些状态机代码了，对了还有一个 carry_image 变量，保持了现在蚂蚁正在搬运物体的图像，或许是一片树叶，或许是一只死蜘蛛。这里我们写了一个加强的 render 函数，因为我们可能还需要画一下搬的东西。

建造大脑

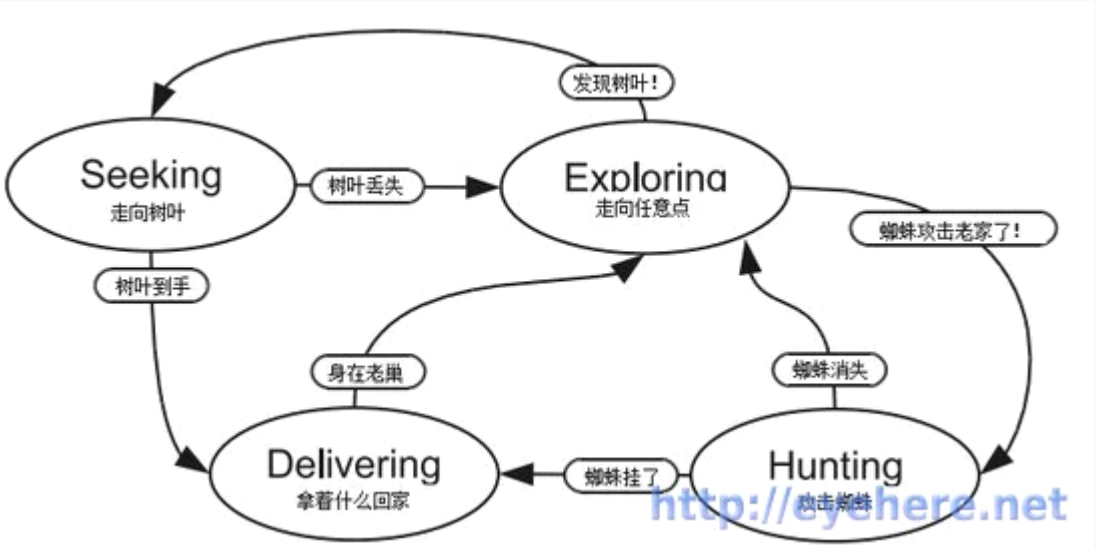
我们给每一只蚂蚁赋予四个状态，这样才能足够建造我们的蚂蚁的状态机。在建造状态机之前，我们得先把这些状态的详细信息列出来。

状态	动作
探索(Exploring)	随机的走向一个点
搜集(Seeking)	向一篇树叶前进
搬运(Dellivering)	搬运一个什么回去
狩猎(Hunting)	攻击一只蜘蛛

我们也需要定义一下各个状态之间的链接，或者可以叫转移条件。这里举两个例子（实际上不止）：

条件	转移状态
发现树叶	搜集
有蜘蛛攻击	狩猎

我们还是最终画一张图来表示整个状态机：



高水平的你也许可以看着上面的图写状态机了，不过为了方便先建立一个 State 类，来保存一个状态。

很简单，只是一个框子，实际上什么都不做：

```
classState():
def__init__(self, name):
self.name=name
defdo_actions(self):
pass
defcheck_conditions(self):
pass
defentry_actions(self):
pass
defexit_actions(self):
pass
```

然后可以建立一个状态机类来管理这些状态，这个状态机可是整个代码的核心类。

```
classStateMachine():
def__init__(self):
self.states={}# 存储状态
self.active_state=None# 当前有效状态
defadd_state(self, state):
```

```
6  # 增加状态
7  self.states[state.name]=state
8  defthink(self):
9  ifself.active_stateisNone:
10 return
11 # 执行有效状态的动作，并做转移检查
12 self.active_state.do_actions()
13 new_state_name=self.active_state.check_conditions()
14 ifnew_state_nameisnotNone:
15 self.set_state(new_state_name)
16 defset_state(self, new_state_name):
17 # 更改状态，执行进入/退出动作
18 ifself.active_stateisnotNone:
19 self.active_state.exit_actions()
20 self.active_state=self.states[new_state_name]
21 self.active_state.entry_actions()
```

然后就可以通过继承 State 创建一系列的实际状态了，这些状态传递给 StateMachine 保留并运行。

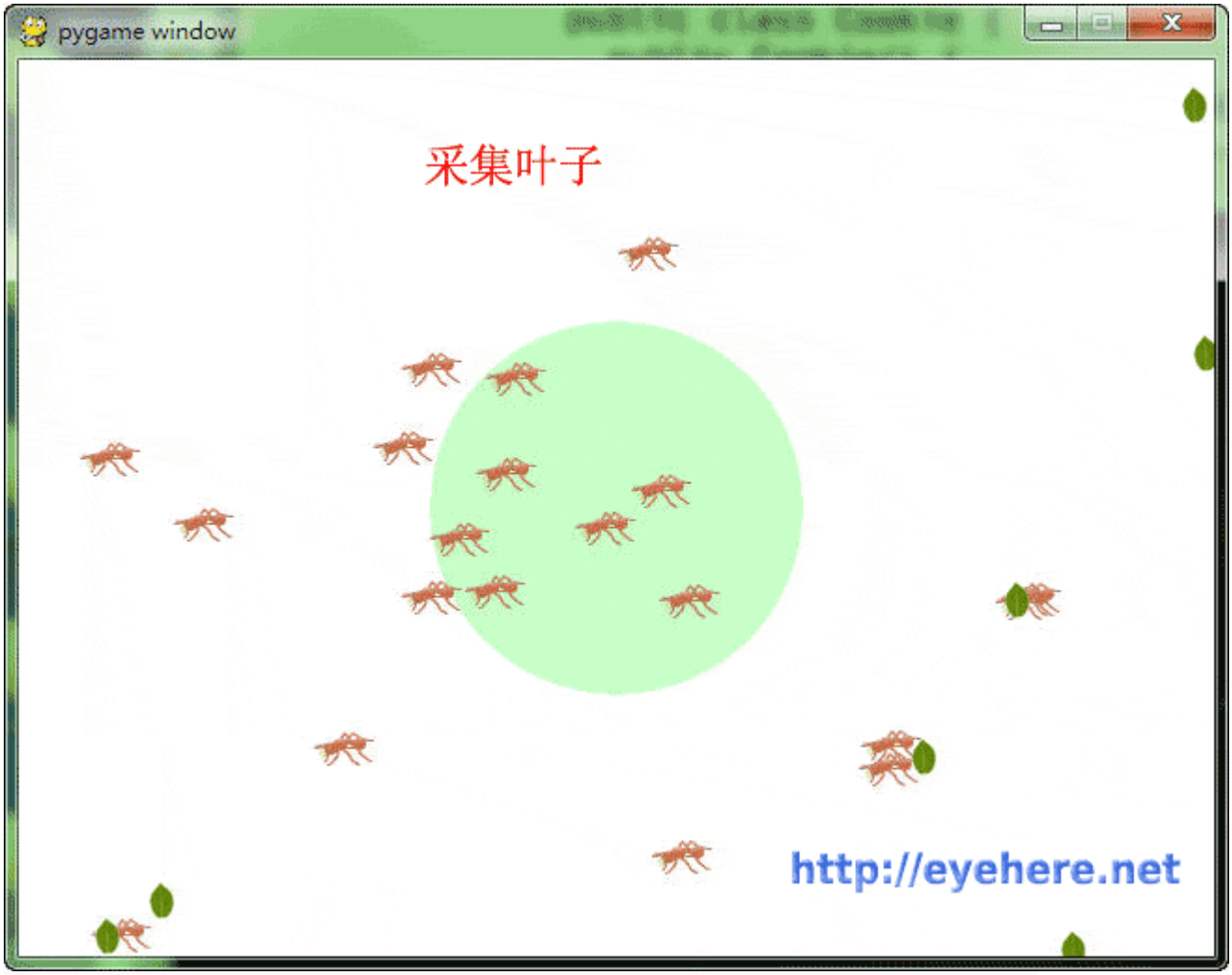
StateMachine 类的 think 方法是检查当前有效状态并执行其动作的，最后还可能会调用 set_state 来进入下一个状态。

我们应该根据上面的四个状态表格建立状态了，有鉴于本次的代码已经很多了，需要好好消化……下一次将会一次性给出完整代码，到时候就能看到一个丰富多彩的昆虫世界了！尽请期待~

用 Python 和 Pygame 写游戏-从入门到精通（16）

星期二, 19. 七月 2011

经历了长年的艰苦卓绝的披星戴月的惨绝人寰的跋山涉水，我们终于接近了 AI 之旅的尾声（好吧，实际上我们这才是刚刚开始），。这一次真正展示一下这几回辛勤工作的结果，最后的画面会是这个样子：



下面给出完整代码（注意需要 gameobjects 库才可以运行，参考之前的向量篇）：

```
SCREEN_SIZE=(640,480)
NEST_POSITION=(320,240)
ANT_COUNT=20
NEST_SIZE=100.
importpygame
frompygame.localsimport*
fromrandomimportrandint, choice
fromgameobjects.vector2importVector2
classState(object):
def__init__(self, name):
self.name=name
defdo_actions(self):
pass
defcheck_conditions(self):
pass
```

```
defentry_actions(self):
pass
defexit_actions(self):
pass
classStateMachine(object):
def__init__(self):
self.states={}
self.active_state=None
defadd_state(self, state):
self.states[state.name]=state
defthink(self):
ifself.active_stateisNone:
return
self.active_state.do_actions()
new_state_name=self.active_state.check_conditions()
ifnew_state_nameisnotNone:
self.set_state(new_state_name)
defset_state(self, new_state_name):
ifself.active_stateisnotNone:
self.active_state.exit_actions()
self.active_state=self.states[new_state_name]
self.active_state.entry_actions()
classWorld(object):
def__init__(self):
self.entities={}
self.entity_id=0
self.background=pygame.surface.Surface(SCREEN_SIZE).convert()
self.background.fill((255,255,255))
pygame.draw.circle(self.background, (200,255,200), NEST_POSITION,int(NEST_SIZE))
defadd_entity(self, entity):
self.entities[self.entity_id]=entity
entity.id=self.entity_id
self.entity_id+=1
defremove_entity(self, entity):
delself.entities[entity.id]
defget(self, entity_id):
ifentity_idinself.entities:
returnself.entities[entity_id]
else:
returnNone
defprocess(self, time_passed):
time_passed_seconds=time_passed/1000.0
forentityinself.entities.values():
entity.process(time_passed_seconds)
```

```
defrender(self, surface):
surface.blit(self.background, (0,0))
forentityinself.entities.itervalues():
entity.render(surface)
defget_close_entity(self, name, location,range=100.):
location=Vector2(*location)
forentityinself.entities.itervalues():
ifentity.name==name:
distance=location.get_distance_to(entity.location)
ifdistance <range:
returnentity
returnNone
classGameEntity(object):
def__init__(self, world, name, image):
self.world=world
self.name=name
self.image=image
self.location=Vector2(0,0)
self.destination=Vector2(0,0)
self.speed=0.
self.brain=StateMachine()
self.id=0
defrender(self, surface):
x, y=self.location
w, h=self.image.get_size()
surface.blit(self.image, (x-w/2, y-h/2))
defprocess(self, time_passed):
self.brain.think()
ifself.speed >0.andself.location !=self.destination:
vec_to_destination=self.destination-self.location
distance_to_destination=vec_to_destination.get_length()
heading=vec_to_destination.get_normalized()
travel_distance=min(distance_to_destination, time_passed*self.speed)
self.location+=travel_distance*heading
classLeaf(GameEntity):
def__init__(self, world, image):
GameEntity.__init__(self, world,"leaf", image)
classSpider(GameEntity):
def__init__(self, world, image):
GameEntity.__init__(self, world,"spider", image)
self.dead_image=pygame.transform.flip(image,0,1)
self.health=25
self.speed=50.+randint(-20,20)
defbitten(self):
```

```
104 self.health-=1
105 ifself.health <=0:
106 self.speed=0.
107 self.image=self.dead_image
108 self.speed=140.
109 defrender(self, surface):
110 GameEntity.render(self, surface)
111 x, y=self.location
112 w, h=self.image.get_size()
113 bar_x=x-12
114 bar_y=y+h/2
115 surface.fill( (255,0,0), (bar_x, bar_y,25,4))
116 surface.fill( (0,255,0), (bar_x, bar_y,self.health,4))
117 defprocess(self, time_passed):
118 x, y=self.location
119 ifx > SCREEN_SIZE[0]+2:
120 self.world.remove_entity(self)
121 return
122 GameEntity.process(self, time_passed)
123 classAnt(GameEntity):
124 def__init__(self, world, image):
125 GameEntity.__init__(self, world,"ant", image)
126 exploring_state=AntStateExploring(self)
127 seeking_state=AntStateSeeking(self)
128 delivering_state=AntStateDelivering(self)
129 hunting_state=AntStateHunting(self)
130 self.brain.add_state(exploring_state)
131 self.brain.add_state(seeking_state)
132 self.brain.add_state(delivering_state)
133 self.brain.add_state(hunting_state)
134 self.carry_image=None
135 defcarry(self, image):
136 self.carry_image=image
137 defdrop(self, surface):
138 ifself.carry_image:
139 x, y=self.location
140 w, h=self.carry_image.get_size()
141 surface.blit(self.carry_image, (x-w, y-h/2))
142 self.carry_image=None
143 defrender(self, surface):
144 GameEntity.render(self, surface)
145 ifself.carry_image:
146 x, y=self.location
147 w, h=self.carry_image.get_size()
```

```
surface.blit(self.carry_image, (x-w, y-h/2))
classAntStateExploring(State):
def__init__(self, ant):
State.__init__(self,"exploring")
self.ant=ant
defrandom_destination(self):
w, h=SCREEN_SIZE
self.ant.destination=Vector2(randint(0, w), randint(0, h))
defdo_actions(self):
ifrandint(1,20)==1:
self.random_destination()
defcheck_conditions(self):
leaf=self.ant.world.get_close_entity("leaf",self.ant.location)
ifleafisnotNone:
self.ant.leaf_id=leaf.id
return"seeking"
spider=self.ant.world.get_close_entity("spider", NEST_POSITION, NEST_SIZE)
ifspiderisnotNone:
ifself.ant.location.get_distance_to(spider.location) <100.:
self.ant.spider_id=spider.id
return"hunting"
returnNone
defentry_actions(self):
self.ant.speed=120.+randint(-30,30)
self.random_destination()
classAntStateSeeking(State):
def__init__(self, ant):
State.__init__(self,"seeking")
self.ant=ant
self.leaf_id=None
defcheck_conditions(self):
leaf=self.ant.world.get(self.ant.leaf_id)
ifleafisNone:
return"exploring"
ifself.ant.location.get_distance_to(leaf.location) <5.0:
self.ant.carry(leaf.image)
self.ant.world.remove_entity(leaf)
return"delivering"
returnNone
defentry_actions(self):
leaf=self.ant.world.get(self.ant.leaf_id)
ifleafisnotNone:
self.ant.destination=leaf.location
self.ant.speed=160.+randint(-20,20)
```

```
classAntStateDelivering(State):
def__init__(self, ant):
State.__init__(self,"delivering")
self.ant=ant
defcheck_conditions(self):
ifVector2(*NEST_POSITION).get_distance_to(self.ant.location) < NEST_SIZE:
if(randint(1,10)==1):
self.ant.drop(self.ant.world.background)
return"exploring"
returnNone
defentry_actions(self):
self.ant.speed=60.
random_offset=Vector2(randint(-20,20), randint(-20,20))
self.ant.destination=Vector2(*NEST_POSITION)+random_offset
classAntStateHunting(State):
def__init__(self, ant):
State.__init__(self,"hunting")
self.ant=ant
self.got_kill=False
defdo_actions(self):
spider=self.ant.world.get(self.ant.spider_id)
ifspiderisNone:
return
self.ant.destination=spider.location
ifself.ant.location.get_distance_to(spider.location) <15.:
ifrandint(1,5)==1:
spider.bitten()
ifspider.health <=0:
self.ant.carry(spider.image)
self.ant.world.remove_entity(spider)
self.got_kill=True
defcheck_conditions(self):
ifself.got_kill:
return"delivering"
spider=self.ant.world.get(self.ant.spider_id)
ifspiderisNone:
return"exploring"
ifspider.location.get_distance_to(NEST_POSITION) > NEST_SIZE*3:
return"exploring"
returnNone
defentry_actions(self):
self.speed=160.+randint(0,50)
defexit_actions(self):
self.got_kill=False
```

```
defrun():
pygame.init()
screen=pygame.display.set_mode(SCREEN_SIZE,0,32)
world=World()
w, h=SCREEN_SIZE
clock=pygame.time.Clock()
ant_image=pygame.image.load("ant.png").convert_alpha()
leaf_image=pygame.image.load("leaf.png").convert_alpha()
spider_image=pygame.image.load("spider.png").convert_alpha()
forant_noinxrange(ANT_COUNT):
ant=Ant(world, ant_image)
ant.location=Vector2(randint(0, w), randint(0, h))
ant.brain.set_state("exploring")
world.add_entity(ant)
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
return
time_passed=clock.tick(30)
ifrandint(1,10)==1:
leaf=Leaf(world, leaf_image)
leaf.location=Vector2(randint(0, w), randint(0, h))
world.add_entity(leaf)
ifrandint(1,100)==1:
spider=Spider(world, spider_image)
spider.location=Vector2(-50, randint(0, h))
spider.destination=Vector2(w+50, randint(0, h))
world.add_entity(spider)
world.process(time_passed)
world.render(screen)
pygame.display.update()
if__name__=="__main__":
run()
```

```
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
```

这个程序的长度超过了以往任何一个，甚至可能比我们写的加起来都要长一些。然而它可以展现给我们的也前所未有的惊喜。无数勤劳的小蚂蚁在整个地图上到处觅食，随机出现的叶子一旦被蚂蚁发现，

就会搬回巢穴，而蜘蛛一旦出现在巢穴范围之内，就会被蚂蚁们群起而攻之，直到被驱逐出地图范围或者挂了，蜘蛛的尸体也会被带入巢穴。

这个代码写的不够漂亮，没有用太高级的语法，甚至都没有注释天哪……基本代码都在前面出现了，只是新引入了四个新的状态，*AntStateExploring*、*AntStateSeeking*、*AntStateDelivering* 和 *AntStateHunting*，意义的话前面已经说明。比如说 AntStateExploring，继承了基本的 Stat，这个状态的动作平时就是让蚂蚁以一个随机的速度走向屏幕随机一个点，在此过程中，*check_conditions* 会不断检查周围的环境，发现了树叶或蜘蛛都会采取相应的措施（进入另外一个状态）。

游戏设计艺术中，创建一个漂亮的 AI 是非常有挑战性也非常有趣的事情。好的 AI 能让玩家沉浸其中，而糟糕的 AI 则让人感到非常乏味（有的时候 AI 中的一些 bug 被当作秘籍使用，也挺有意思的，不过如果到处是“秘籍”，可就惨了）。而且，AI 是否足够聪明有时候并不与代码量直接相关，看看我们这个演示，感觉上去蚂蚁会合作攻击蜘蛛，而实际上它们都是独立行动的，不过就结果而言蚂蚁们看起来都很聪明。

对 AI 而已，状态机是个很有力的工具（当然状态机不仅仅用在这里），因为状态机可以把复杂的系统分割成几个容易实现的小段。而这每一小部分都是对一些简单思考或动作的模拟，即便不是那么容易转化为代码，也很容易模拟。在游戏中，我们只需要模拟就足够了。

我们这几次讲述的东西相当有用，尽管不是那么直观，但对于游戏设计至关重要，而此次的蚁巢演示，也给我们揭示了 AI 系统的种种，至少这个系统式可以运作的了，不错不错~ 参天大树也是从小树苗开始的。

本次使用的图像资源：

叶子：leaf.png

蚂蚁：ant.png

蜘蛛：spider.png

下一次，我们开始更加激动人心的项目，3D 图像！

用 Python 和 Pygame 写游戏-从入门到精通（17）

星期六, 6. 八月 2011

最近有些忙，没有更新这个系列，不行啊不行，抓紧更新一篇，这几次可是3D 啊3D，多么诱人的词啊……

游戏通常希望营造一个真实的世界，越接近真实越好啊，这样的代入感会很强。在早期，由于硬件的限制，游戏只能提供一些2D 的图像，因为这对于电脑绘图是最容易的。还好随着技术发展，现在的显卡已经可以画出很逼真的3D 画面了，所以“硬件杀手”游戏层出不穷，贫困游戏迷的噩梦啊。

在开开心心的继续之前，是不是有记忆力好的人想起这个系列第一篇里面我说过，pygame 不适合做3D，怎么这里又厚颜无耻的开始说3D 了？这不是搬石头砸自己的脚么：）这里我要仔细说明一下，所谓3D，说到底就是利用透视原理，在2D 的画面上创造出有纵深错觉（说白了也就是近大远小）的画面而已，毕竟，屏幕是平的，怎么可能真的画出距离呢？换句话说，计算机3D 的本质还是2D，只不过额外多了很多东西。

在纯 pygame 中，我们画3D 画面就是通过计算在2D 图像上画一些大小不一的东西 ）

距离的魔法

我们看现实中的东西，和我们看画面上的东西，最大差别在于能感受现实物体的距离。而距离的产生，则是因为我们双眼看到的东西是不同的，两眼交替闭合，你会发现眼前的东西左右移动。一只眼睛则很难正确的判断距离，虽然比上眼睛还是能感觉到远近，但更精细一点，比如很难把线穿过针眼。

我们在3D 画面上绘图的时候，就要遵循这个规律，看看下面的代码。

```
importpygame
frompygame.localsimport*
fromrandomimportrandint
classStar(object):
def__init__(self, x, y, speed):
self.x=x
self.y=y
self.speed=speed
defrun():
pygame.init()
screen=pygame.display.set_mode((640,480))#, FULLSCREEN)
stars=[]
# 在第一帧，画上一些星星
forninxrange(200):
x=float(randint(0,639))
y=float(randint(0,479))
speed=float(randint(10,300))
stars.append( Star(x, y, speed) )
clock=pygame.time.Clock()
white=(255,255,255)
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
return
ifevent.type==KEYDOWN:
return
# 增加一颗新的星星
y=float(randint(0,479))
speed=float(randint(10,300))
star=Star(640., y, speed)
stars.append(star)
time_passed=clock.tick()
```

```
time_passed_seconds=time_passed/1000.
screen.fill((0,0,0))
# 绘制所有的星
forstarinstars:
new_x=star.x-time_passed_seconds*star.speed
pygame.draw.aaline(screen, white, (new_x, star.y), (star.x+1., star.y))
star.x=new_x
defon_screen(star):
returnstar.x >0
# 星星跑出了画面，就删了它
stars=filter(on_screen, stars)
pygame.display.update()
if__name__=="__main__":
run()
```

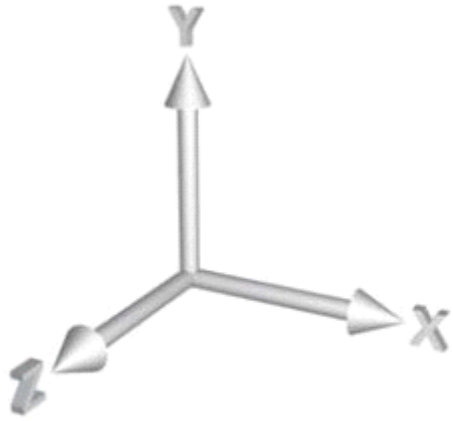
这里你还可以把 FULLSCREEN 加上，更有感觉。

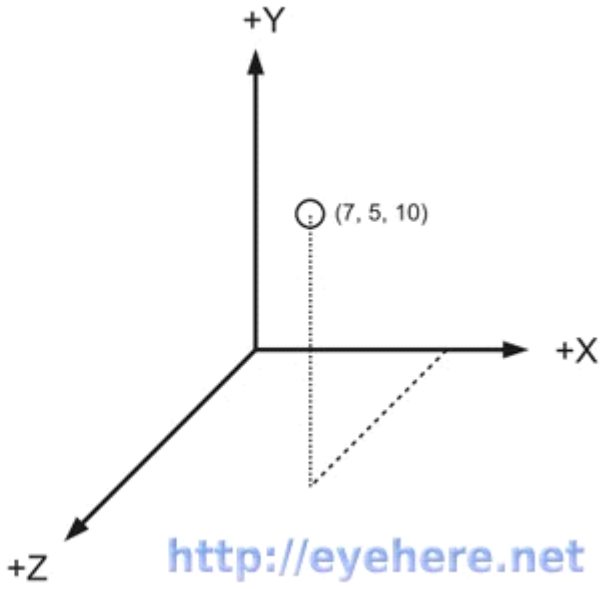
这个程序给我的画面，发挥一下你的想象，不是一片宇宙么，无数的星云穿梭，近的速度更快，远的则很慢。而实际上看代码，我们只是画了一些长短不同的线而已！虽然很简单，还是用了不少不少 python 的技术，特别是函数式编程的（小）技巧。不过强大的你一定没问题）但是 pygame 的代码，没有任何没讲过的，为什么这样就能有3D 的效果了？感谢你的大脑，因为它知道远的看起来更

慢，所以这样的错觉就产生了。

理解3D 空间

3D 空间的事情，基本就是立体几何的问题，高中学一半应该就差不多理解了，这里不多讲了。你能明白下图的小球在(7, 5, 10)的位置，换句话说，如果你站在原点，面朝 Z 轴方向。那么小球就在你左边7，上面5，前面10的位置。这就够了~





使用3D 向量

我们已经学习了二维向量来表达运动，在三维空间内，当然要使用三维的向量。其实和二维的概念都一样，加减缩放啥的，这里就不用三个元素的元组列表先演练一番了，直接祭出我们的 gameobjects 神器吧！

```
fromgameobjects.vector3import*
A=Vector3(6,8,12)
B=Vector3(10,16,12)
print"A is", A
print"B is", B
print"Magnitude of A is", A.get_magnitude()
print"A+B is", A+B
print"A-B is", A-B
print"A normalized is", A.get_normalized()
```

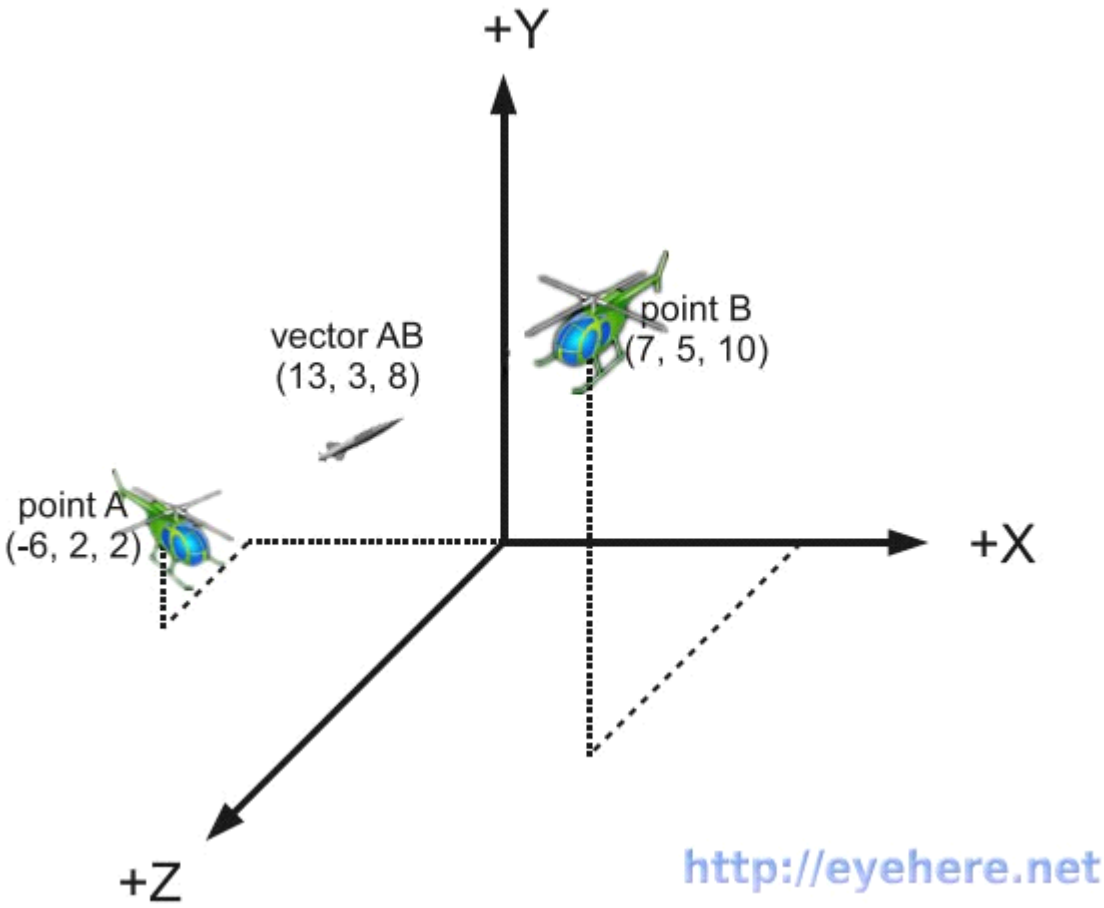
```
10 print"A*2 is", A*2
```

运行一下看看结果吧，有些无趣？确实，光数字没法展现3D 的美啊，下一次，让我们把物体在立体空间内运动起来。

3D 是非常酷的技术，同时也就意味着更多的工作，上次的简单介绍之后，这次还要讲更多2D 到3D 的新概念。

基于时间的三维移动

我们使用 Vector3类来进行3D 上的移动，与2D 非常类似，看下面一个例子：



直升机 A 在(-6, 2, 2)的位置上，目标是直升机 B(7, 5, 10)，A 想摧毁 B，所以发射了一枚火箭 AB，现在我们得把火箭的运动轨迹过程给画出来，否则一点发射敌机就炸了，多没意思啊~~ 通过计算出两者之间的向量为(13, 3, 8)，然后单位化这个向量，这样就可以在运动中用到了，下面的代码做了这些事情。

```
fromgameobjects.vector3import*
A=(-6,2,2)
B=(7,5,10)
plasma_speed=100.# meters per second
AB=Vector3.from_points(A, B)
print"Vector to droid is", AB
distance_to_target=AB.get_magnitude()
print"Distance to droid is", distance_to_target,"meters"
plasma_heading=AB.get_normalized()
print"Heading is", plasma_heading
#######输出结果#########
Vector to droidis(13,3,8)
Distance to droidis15.5563491861meters
```

```
14 Headingis(0.835672,0.192847,0.514259)
```

然后不停的重绘火箭的位置，用这个语句：

rocket_location += heading * time_passed_seconds * speed

不过我们还不能直接在 pygame 中绘制3D 物体，得先学习一下下面讲的，“如何把3D 转换为2D”。

3D 透视

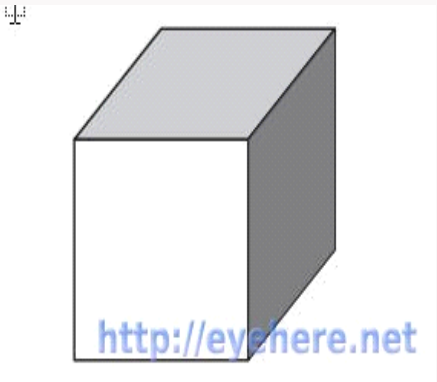
如果您初中美术认真学了的话，应该会知道这里要讲什么，还记得当初我们是如何在纸上画立方体的？

忘了？OK，从头开始说起吧，存储、计算3D 坐标是非常容易的，但是要把它展现到屏幕上就不那么简单了，因为 pygame 中所有的绘图函数都只接受2D 坐标，因此，我们必须把这些3D 的坐标**投影**到2D 的图面上。

平行投影

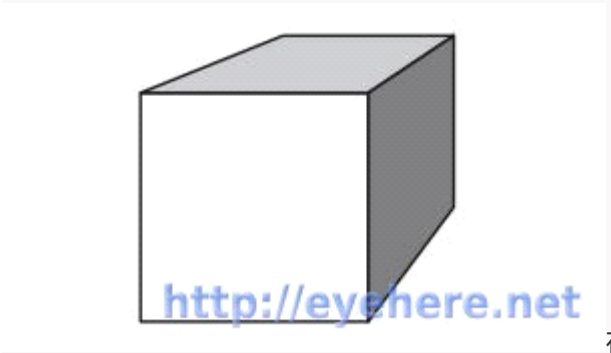
最简单的投影方法是——把第三个坐标 z 坐标给丢弃，用这样的一个简单的函数就可以做到：

```
1 defparallel_project(vector3):
2 return(vector3.x, vector3.y)
```



尽管这样的转换简单又快速，我们却不能用它。为什么？效果太糟糕了，我们甚至无法在最终的画面上感受到一点立体的影子，这个世界看起来还是平的，没有那个物体看起来比其他物体更远或更近。就好像我右边这幅图一样。

立体投影



在3D游戏中使用的更为广泛且合理的技术是立体投影，因为它的结果更为真实。立体投影把远处的物体缩小了，也就是使用透视法（foreshortening），如左图所示，然后下面是我们的转换函数，看起来也很简单：

```
defperspective_project(vector3, d):
x, y, z=vector3
return(x*d/z, -y*d/z)
```

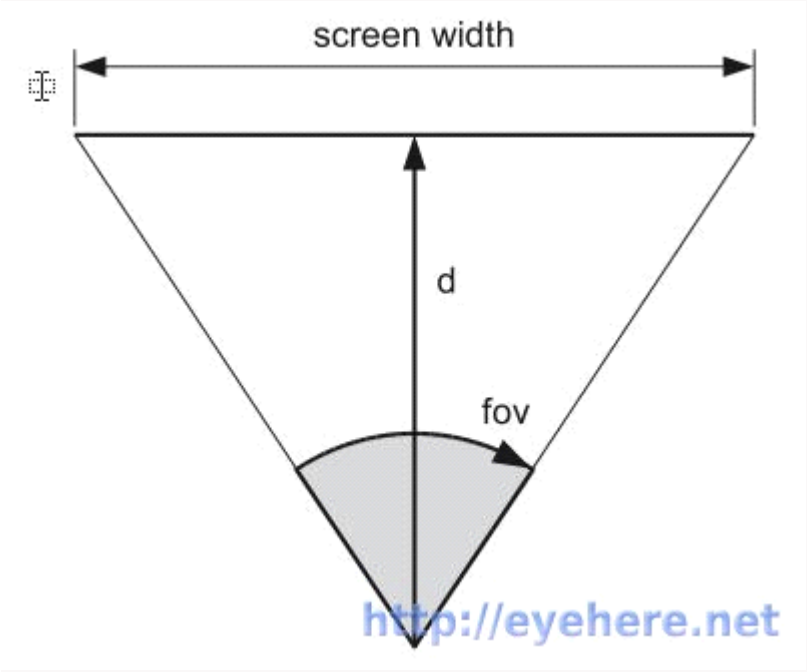
与上一个转换函数不同的是，这个转换函数还接受一个 d 参数（后面讨论），然后所有的 x、y 坐标都会接受这个 d 的洗礼，同时 z 也会插一脚，把原本的坐标进行缩放。

d 的意思是**视距** (viewing distance)，也就是摄像头到3D 世界物体在屏幕上的像素体现之间的距离。比如说，一个在(10, 5, 100)的物体移动到了(11, 5, 100)，视距是100的时候，它在屏幕上就刚好移动了1个像素，但如果它的 z 不是100，或者视距不是100，那么可能移动距离就不再是1个像素的距离。有些抽象，不过玩过3D 游戏的话（这里指国外的3D 大作），都有一种滚轮调节远近的功能，这就是视距（当然调的时候视野也会变化，这个下面说）。

在我们玩游戏的时候，视距就为我们的眼睛到屏幕的直线距离（以像素为单位）。

视野

那么我们怎么选取一个好的 d 呢？我们当然可以不断调整实验来得到一个，不过我们还可以通过**视野** (field of view)来计算一个出来。视野也就是在一个时刻能看到的角度。看一下左图的视野和视距的关系，可以看到两者是有制约关系，当视野角度（fov）增大的时候，d 就会减小；而 d 增加的话，视野角度就会减小，能看到的东西也就变少了。



视野是决定在3D 画面上展现多少东西的绝好武器，然后我们还需要一个 d 来决定透视深度，使用一点点三角只是，我们就可以从 fov 计算出 d，写一下下面的代码学习学习：

在 Internet 上，你总是能找到99%以上的需要的别人写好的代码。不过偶尔还是要自己写一下的，不用担心自己的数学是不及格的，这个很简单~ 很多时候实际动手试一下，你就能明白更多。

```
frommathimporttan
defcalculate_viewing_distance(fov, screen_width):
d=(screen_width/2.0)/tan(fov/2.0)
returnd
```

fov 角度可能取45~60°比较合适，这样看起来很自然。当然每个人每个游戏都有特别的地方，比如 FPS 的话，fov 可能比较大；而策略性游戏的 fov 角度会比较小，这样可以看到更多的东西。很多时候还需要不停的变化 fov，最明显的 CS 中的狙击枪（从没玩过，不过听过），开出来和关掉是完全不同的效果，改的就是视野角度。

今天又是补充了一大堆知识，等不及了吧~我们下一章就能看到一个用 pygame 画就的3D 世界了！

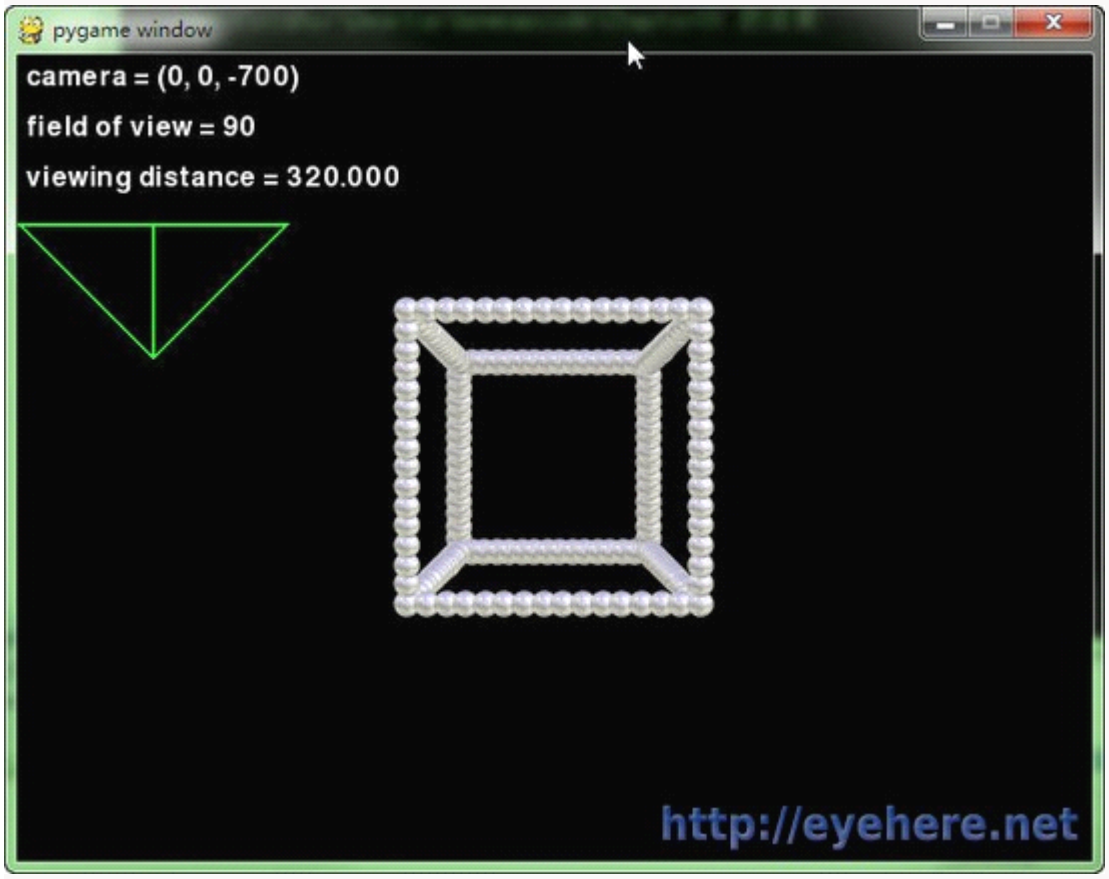
用 Python 和 Pygame 写游戏-从入门到精通（19）

星期四, 11. 八月 2011

3D 世界

让我们现在开始写一个3D 的程序，巩固一下这几次学习的东西。因为我们还没有好好深入如何画3D 物体，暂时就先用最简单的投影（上次讨论过的第二种）方法来画吧。这个程序画一个空间里的立方体，只不过各个部分并不会随着距离而产生大小上的变化。

您可以看到，很多的小球构成了立方体的各个边，通过按住方向键，可以水平或垂直方向的更改“摄像头”的位置，Q 和 A 键会把摄像头拉近或拉远，而 W 和 S 会改变视距，绿色的三角是视距和视角的示意图。fov 角大的话，立方体就显得比较短，反之就显得比较长。



代码稍微有点长，下面有解释，静下心来慢慢阅读。

```
importpygame
frompygame.localsimport*
fromgameobjects.vector3importVector3
frommathimport*
fromrandomimportrandint
```

```
SCREEN_SIZE=(640,480)
CUBE_SIZE=300
defcalculate_viewing_distance(fov, screen_width):
d=(screen_width/2.0)/tan(fov/2.0)
returnd
defrun():
pygame.init()
screen=pygame.display.set_mode(SCREEN_SIZE,0)
default_font=pygame.font.get_default_font()
font=pygame.font.SysFont(default_font,24)
ball=pygame.image.load("ball.png").convert_alpha()
# 3D points
points=[]
fov=90.# Field of view
viewing_distance=calculate_viewing_distance(radians(fov), SCREEN_SIZE[0])
# 边沿的一系列点
forxinxrange(0, CUBE_SIZE+1,20):
edge_x=x==0orx==CUBE_SIZE
foryinxrange(0, CUBE_SIZE+1,20):
edge_y=y==0ory==CUBE_SIZE
forzinxrange(0, CUBE_SIZE+1,20):
edge_z=z==0orz==CUBE_SIZE
ifsum((edge_x, edge_y, edge_z)) >=2:
point_x=float(x)-CUBE_SIZE/2
point_y=float(y)-CUBE_SIZE/2
point_z=float(z)-CUBE_SIZE/2
points.append(Vector3(point_x, point_y, point_z))
# 以 z 序存储，类似于 css 中的 z-index
defpoint_z(point):
returnpoint.z
points.sort(key=point_z, reverse=True)
center_x, center_y=SCREEN_SIZE
center_x/=2
center_y/=2
ball_w, ball_h=ball.get_size()
ball_center_x=ball_w/2
ball_center_y=ball_h/2
camera_position=Vector3(0.0,0.0,-700.)
camera_speed=Vector3(300.0,300.0,300.0)
clock=pygame.time.Clock()
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
return
```

```
screen.fill((0,0,0))
pressed_keys=pygame.key.get_pressed()
time_passed=clock.tick()
time_passed_seconds=time_passed/1000.
direction=Vector3()
ifpressed_keys[K_LEFT]:
direction.x=-1.0
elifpressed_keys[K_RIGHT]:
direction.x=+1.0
ifpressed_keys[K_UP]:
direction.y=+1.0
elifpressed_keys[K_DOWN]:
direction.y=-1.0
ifpressed_keys[K_q]:
direction.z=+1.0
elifpressed_keys[K_a]:
direction.z=-1.0
ifpressed_keys[K_w]:
fov=min(179., fov+1.)
w=SCREEN_SIZE[0]
viewing_distance=calculate_viewing_distance(radians(fov), w)
elifpressed_keys[K_s]:
fov=max(1., fov-1.)
w=SCREEN_SIZE[0]
viewing_distance=calculate_viewing_distance(radians(fov), w)
camera_position+=direction*camera_speed*time_passed_seconds
# 绘制点
forpointinpoints:
x, y, z=point-camera_position
ifz >0:
x=x*viewing_distance/z
y=-y*viewing_distance/z
x+=center_x
y+=center_y
screen.blit(ball, (x-ball_center_x, y-ball_center_y))
# 绘制表
diagram_width=SCREEN_SIZE[0]/4
col=(50,255,50)
diagram_points=[]
diagram_points.append( (diagram_width/2,100+viewing_distance/4) )
diagram_points.append( (0,100) )
diagram_points.append( (diagram_width,100) )
diagram_points.append( (diagram_width/2,100+viewing_distance/4) )
diagram_points.append( (diagram_width/2,100) )
```

```
pygame.draw.lines(screen, col,False, diagram_points,2)
# 绘制文字
white=(255,255,255)
cam_text=font.render("camera = "+str(camera_position),True, white)
screen.blit(cam_text, (5,5))
fov_text=font.render("field of view = %i"%int(fov),True, white)
screen.blit(fov_text, (5,35))
txt="viewing distance = %.3f"%viewing_distance
d_text=font.render(txt,True, white)
screen.blit(d_text, (5,65))
pygame.display.update()
if__name__=="__main__":
run()
```

```
138
139
140
141
142
```

上面的例子使用 Vector3来管理向量数据，点的存储是按照 z 坐标来的，这样在 blit 的时候，离摄像机近的后画，就可以覆盖远的，否则看起来就太奇怪了……

在主循环的代码中，会根据按键摄像头会更改位置——当然这是用户的感觉，实际上代码做的是更改了点的位置。而3D 的移动和2D 是非常像的，只不过多了一个 z 来判断覆盖远近（也就是绘制顺序），一样的基于时间移动，一样的向量运算，只是由 Vector2变为了 Vector3。

然后代码需要绘制全部的点。首先，点的位置需要根据 camera_position 变量校正，如果结果的 z 比0还大，说明点在摄像头之前，需要画的，否则就是不需要画。y 需要校准一下方向，最后把 x、y 定位在中间（小球还是有一点点尺寸的）。

留下的代码是给出信息的，都是我们学习过的东西了。

如果想好好学习，把立方体换成其他的图像吧！改一下更能加深印象。

3D 第一部分总结

3D 是迄今为止游戏发展中最大的里程碑（下一个会是什么呢？虚拟体验？），我们这几次学习的，是3D 的基础，你可以看到，仅有2D 绘图经验也能很方便的过渡过来。仅仅是 Vector2→Vector3，担任3D 向量还是有一些特有的操作的，需要慢慢学习，但是基本的思想不会变。

但是，请继续思考~ 难道所有的3D 游戏就是一系列的3D 坐标再手动转换为2D 画上去就好了？很可惜或者说很幸运不是的，我们有**3D 引擎**来做这些事情，对 Pygame 来说，原生的3D 处理时不存在的，那么如何真正绘制3D 画面？有一个非常广泛的选择——OpenGL，不了解的请自行 Wiki，不过 OpenGL 并不是 Pygame 的一部分，而且3D 实际做下去实在很繁杂，这个系列就暂时不深入讲解了。

尽管有3D 引擎帮我们做投影等事情，我们这几次学的东西绝对不是白费，对基础的一定了解可以更

好的写入3D游戏，对画面的掌握也会更好。如果有需要，这个系列的主线完成后，会根据大家的要求追加讲解 OpenGL 的内容。

接下来开始讲解 Pygame 中的声音，基本游戏制作的全部知识我们都快学习完了 ）

声音是游戏中必要的元素之一，音效可以给予用户良好的反馈体验。赛车的时候可以听到振奋人心的启动时的引擎声和刹车时轮胎摩擦声，射击游戏中枪支弹药的音效和呐喊助威的嗓音，无一不是让人热血沸腾的要因。

宛若电影，最初的电影史无声的，而自从1927年第一部公认的有声电影放映之后，人们的娱乐项目一下子丰富了好多；游戏中也是啊，好的配音绝对可以给我们的作品增色不少。这几次就是给我们的 pygame 作品中增加美妙的声音的。

什么是声音？

又要开始讲原理了啊，做游戏真是什么都要懂，物理数学美术心理学和编程等等等等，大家都不容易呀~~

声音的本质是振动，通过各种介质传播到我们的耳朵里。基本任何物质都可以振动，比如说一旦我们敲打桌子，桌子表面会快速振动，推动附近的空气一起振动，而这种振动会传播（宛如水中扔一颗石子，水波会慢慢传播一样），这种振动最终进入我们的耳道，使得鼓膜振动，引起我们的听觉。

振动的幅度（**响度**）越大，听到的声音也就越大，这个很好理解，我们越用力拍桌子，声音也就越大（同时手也越疼——）。同时，振动的快慢（**音调**）也会直接影响我们对声音高低的判断，也就是平时说的高音和低音的差别，决定着个音调的要素每秒振动的次数，也就是频率，单位是赫兹(Hz)。比如100Hz 意味着这个振动在1秒内进行了100次。音色也是一个重要指标，敲打木头和金属听到的声音完全不同，是**音色**的作用，这个的要素是有振动波形的形状来决定。

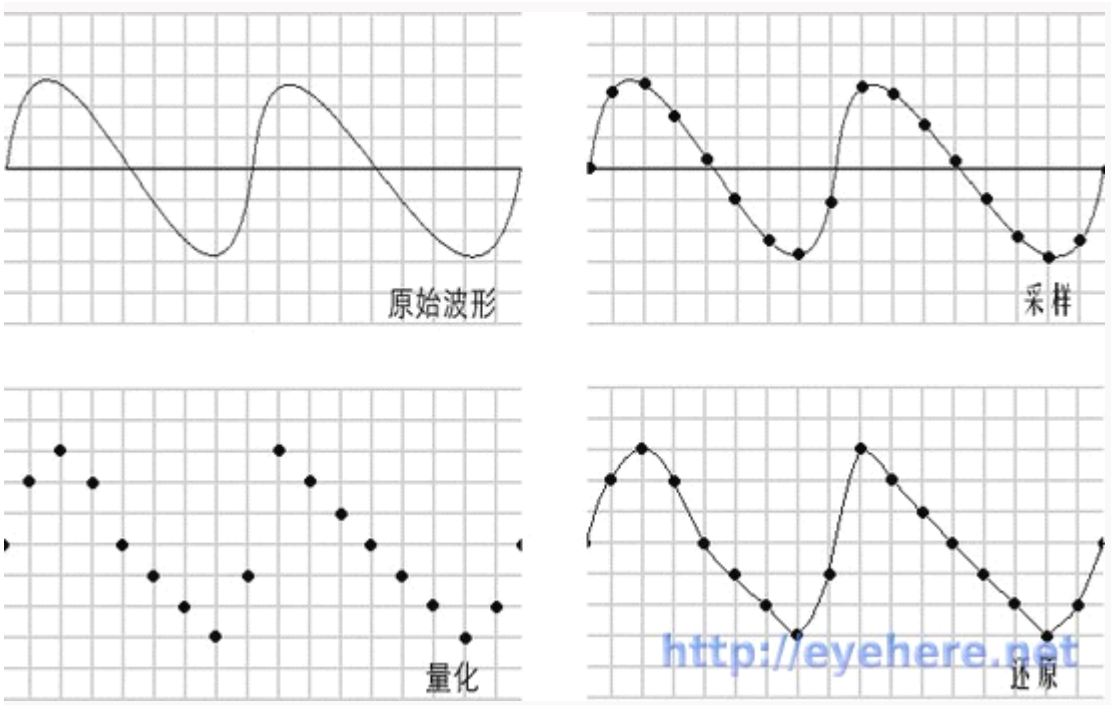
现实中很多声音都是许多不同的声音组合而来的。同时声音在传播的时候也会发生变化，最直接的就是随着距离增大，响度会减小；而在不同的环境中，因为反射和混合，声音的效果也完全不一样。这些都要好好考虑，比如脚步声，空旷的山谷中应该是“空谷足音”的效果，楼梯上则是比较短但是渐渐靠近的效果。甚至发声物体的速度也会影响我们听到的声音，谓之“多普勒效应”……好麻烦！不过最后游戏里可能不是那么符合现实的，比如说太空中发射导弹什么，按说是听不到声音的，因为没有介质传播，不过为了效果好，咱也不在意了……

声音的存储

声音完全是一种模拟的信号，而我们的计算机只能存储数字（二进制）信号，咋办？数字化咯~

（一下说明摘录修改自轩辕天数-丝竹的文章，表示感谢）

以最常见的 WAV 文件为例，要把声音记录成 WAV 格式，电脑要先把声音的波形“画在一张坐标纸上”。然后呢，电脑要看了“横坐标第一格处，波形图的纵坐标是多少啊？哦，差不多是500啊（仅仅是打比方，而且这个“差不多”很关键），那么横坐标第二格呢？...”最后，电脑就得出来一大堆坐标值。然后再经过一些其他后续工作，电脑就把这些坐标值保存下来了。



当要放音的时候，电脑要根据这些“坐标值在坐标纸上面画点”，最后“用线把点连起来”，差不多就把原先的波形还原出来了。其实数字化录音基本上就是这样的原理。

电脑记录波形时，用的坐标纸格子越密，自然记录下来的点就越多、越精确，将来还原出来的波形就越接近原始波形？上边例子的横坐标轴格子密度就相当于采样频率（比如，44.1KHz），纵坐标格子密度就相当于量化精度（比如，16BIT）。这就是“KHZ”、“BIT”的值越高，音乐的音质越好的原因。

这个世界上自然不仅仅有 WAV 这一种存储声音的文件格式，宛若图像文件格式中的 BMP 一样，WAV 是一种无压缩的格式，体积最大；而 OGG 则好像 PNG，是无损的压缩，可以完全保持图像的本真，但是大小又比较小；常用的 MP3，则是类似于 JPG 的有损压缩格式。

声音处理

想要获得声音，最简单的自然是录制，不过有的时候比较困难，比如录制心跳要很高昂的仪器，而录制火山爆发的声音实在过于……

这时候我们可以手动合成声音，而录制获得的声音还需要经过处理，比如净化等，有很多软件可以选

择，开源的 Audacity 就是一个很不错的选择。具体的这里就不说了，一门大学问啊。

网上也有很多声音素材可供下载，好的专业的素材都是卖钱的，哎这个世界什么都是钱啊~~

Pygame 中声音的初始化

这次来不及举一个实际例子放声音了，先说一下初始化。

在 pygame 中，使用 mixer 模块来播放声音，不过在实际播放之前，我们需要使用 pygame.mixer.init 函数来初始化一些参数，不过在有的平台上，pygame.mixer.init 会随着 pygame.init 一起被初始化，pygame 干脆提供了一个 pygame.mixer.pre_init()来进行最先的初始化工作，参数说明如下：

- frequency – 声音文件的采样率，尽管高采样率可能会降低性能，但是再次的声卡都可以轻松对应44.1KHz 的声音回放，所以就设这个值吧；
- size – 量化精度
- stereo – 立体声效果，1：mono，2：stereo，具体请 google，一般设2好了
- buffer – 缓冲大小，2的倍数，设4096就差不多了吧

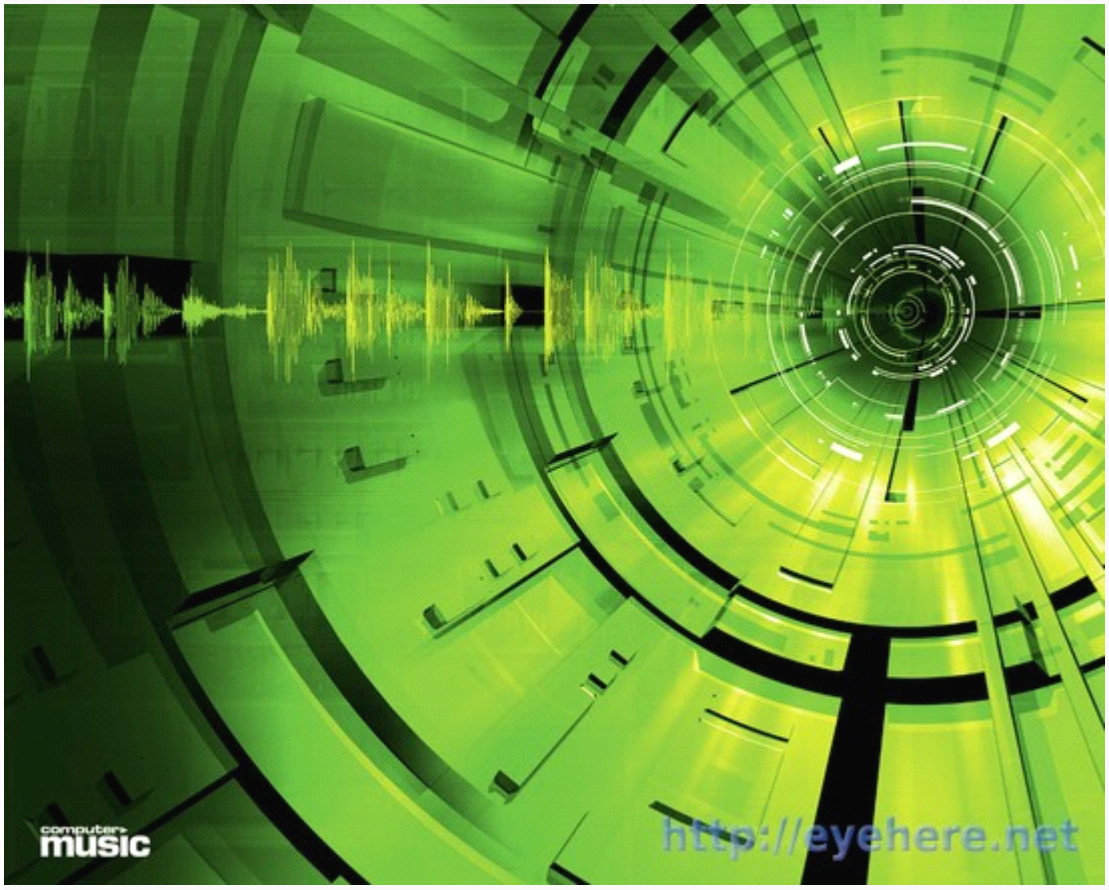
你可以像这样初始化声音：

```
pygame.mixer.pre_init(44100,16,2,4096)
pygame.init()
```

这里先用 pre_init 来设定了参数，然后在 pygame.init 中初始化所有的东西。

如果你需要重新设定声音的参数，那么你需要先执行 pygame.mixer.quit 然后再执行 pygame.mixer.init，不过一般用不到吧……

紧接着上一次，我们继续来看如何在 Pygame 中使用声音。



Sound 对象

在初始化声音系统之后，我们就可以读取一个音乐文件到一个 Sound 对象中了。pygame.mixer.Sound()接受一个文件名，或者也可以使一个文件对象，不过这个文件必须是 WAV 或者 OGG，切记！

```
1 hello_sound=Pygame.mixer.Sound("hello.ogg")
```

一旦这个 Sound 对象出来了，你可以使用 play 方法来播放它。play(loop, maxtime)可以接受两个参数，loop 自然就是重复的次数，-1意味着无限循环，1呢？是**两次**，记住是**重复**的次数而不是播放的次数；maxtime 是指多少毫秒后结束，这个很简单。当你不使用任何参数调用的时候，意味着把这个声音播放一次。一旦 play 方法调用成功，就会返回一个 Channel 对象，否则返回一个 None。

Channel 对象

Channel，也就是声道，可以被声卡混合（共同）播放的数据流。游戏中可以同时播放的声音应该是

有限的，pygame 中默认是8个，你可以通过 pygame.mixer.get_num_channels()来得知当前系统可以同时播放的声道数，而一旦超过，调用 sound 对象的 play 方法就会返回一个 None，如果你确定自己要同时播放很多声音，可以用 set_num_channels()来设定一下，最好一开始就设，因为重新设定会停止当前播放的声音。

那么 Channel 对象有什么用呢？如果你只是想简单的播放一下声音，那么根本不用管这个东西，不过如果你想创造出一点有意境的声音，比如说一辆火车从左到右呼啸而过，那么应该是一开始左声道比较响，然后相当，最后右声道比较响，直至慢慢消失。这就是 Channel 能做到的事情。Channel 的 set_volume(left, right)方法接受两个参数，分别是代表左声道和右声道的音量的小数，从0.0~1.0。

竖起我们的耳朵

OK，来个例子试试吧~

这个例子里我们通过释放金属小球并让它们自由落体和弹跳，听碰撞的声音。这里面不仅仅有这次学习的声音，还有几个一起没有接触到的技术，最重要的一个就是重力的模拟，我们可以设置一个重力因子来影响小球下落的速度，还有一个弹力系数，来决定每次弹跳损失的能量，虽然不难，但是游戏中引入这个东西能让我们的游戏仿真度大大提高。

```
SCREEN_SIZE=(640,480)
# 重力因子，实际上是单位 像素/平方秒
GRAVITY=250.0
# 弹力系数，不要超过1!
BOUNCINESS=0.7
importpygame
frompygame.localsimport*
fromrandomimportrandint
fromgameobjects.vector2importVector2
defstero_pan(x_coord, screen_width):
"""这个函数根据位置决定要播放声音左右声道的音量"""
right_volume=float(x_coord)/screen_width
left_volume=1.0-right_volume
return(left_volume, right_volume)
classBall():
"""小球类，实际上我们可以使用 Sprite 类来简化"""
```

```
def__init__(self, position, speed, image, bounce_sound):
self.position=Vector2(position)
self.speed=Vector2(speed)
self.image=image
self.bounce_sound=bounce_sound
self.age=0.0
defupdate(self, time_passed):
w, h=self.image.get_size()
screen_width, screen_height=SCREEN_SIZE
x, y=self.position
x-=w/2
y-=h/2
# 是不是要反弹了
bounce=False
# 小球碰壁了么？
ify+h >=screen_height:
self.speed.y=-self.speed.y*BOUNCINESS
self.position.y=screen_height-h/2.0-1.0
bounce=True
ifx <=0:
self.speed.x=-self.speed.x*BOUNCINESS
self.position.x=w/2.0+1
bounce=True
elifx+w >=screen_width:
self.speed.x=-self.speed.x*BOUNCINESS
self.position.x=screen_width-w/2.0-1
bounce=True
# 根据时间计算现在的位置，物理好的立刻发现这其实不标准，
# 正规的应该是“s = 1/2*g*t*t”，不过这样省事省时一点，咱只是模拟~
self.position+=self.speed*time_passed
# 根据重力计算速度
self.speed.y+=time_passed*GRAVITY
ifbounce:
self.play_bounce_sound()
self.age+=time_passed
defplay_bounce_sound(self):
"""这个就是播放声音的函数"""
channel=self.bounce_sound.play()
ifchannelisnotNone:
# 设置左右声道的音量
left, right=stero_pan(self.position.x, SCREEN_SIZE[0])
channel.set_volume(left, right)
defrender(self, surface):
# 真有点麻烦了，有爱的，自己用 Sprite 改写下吧……
```

```
w, h=self.image.get_size()
x, y=self.position
x-=w/2
y-=h/2
surface.blit(self.image, (x, y))
defrun():
# 上一次的内容
pygame.mixer.pre_init(44100,16,2,1024*4)
pygame.init()
pygame.mixer.set_num_channels(8)
screen=pygame.display.set_mode(SCREEN_SIZE,0)
pygame.mouse.set_visible(False)
clock=pygame.time.Clock()
ball_image=pygame.image.load("ball.png").convert_alpha()
mouse_image=pygame.image.load("mousecursor.png").convert_alpha()
# 加载声音文件
bounce_sound=pygame.mixer.Sound("bounce.ogg")
balls=[]
whileTrue:
foreventinpygame.event.get():
ifevent.type==QUIT:
return
ifevent.type==MOUSEBUTTONDOWN:
# 刚刚出来的小球，给一个随机的速度
random_speed=( randint(-400,400), randint(-300,0) )
new_ball=Ball( event.pos,
random_speed,
ball_image,
bounce_sound )
balls.append(new_ball)
time_passed_seconds=clock.tick()/1000.
screen.fill((255,255,255))
# 为防止小球太多，把超过寿命的小球加入这个“死亡名单”
dead_balls=[]
forballinballs:
ball.update(time_passed_seconds)
ball.render(screen)
# 每个小球的的寿命是10秒
ifball.age >10.0:
dead_balls.append(ball)
forballindead_balls:
balls.remove(ball)
mouse_pos=pygame.mouse.get_pos()
screen.blit(mouse_image, mouse_pos)
```

```
105 pygame.display.update()
106 if__name__=="__main__":
107 run()
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
```

这么久了，咱们的游戏终于有了声音，太棒了！不过，是不是游戏的背景音乐也是用 mixer 来播放的呢？这不是一个好主意，因为背景音乐往往很长，比较占资源，pygame 中另外提供了一个 pygame.mixer.music 类来控制背景音乐的播放，这也是我们下一次要讲的内容。

辛苦啦~ 这次是我们系统的 pygame 理论学习的最后一章了，把这次的音乐播放讲完了，pygame 的基础知识就全部 OK 了。不过作为完整的教程，只有理论讲解太过枯燥了，我随后还会加一个或更多的实践篇系列，看需要可能也会追加真3D 等额外的内容。

就像上次所说的，pygame.mixer 并不适合播放长时间的音乐播放，我们要使用 pygame.mixer.music。

pygame.mixer.music 用来播放 MP3和 OGG 音乐文件，不过 MP3并不是所有的系统都支持（Linux 默认就不支持 MP3播放），所以最好还是都用 Ogg 文件，我们可以很容易把 MP3转换为 Ogg 文件，自己搜一下吧。

我们使用 pygame.mixer.music.load()来加载一个文件，然后使用 pygame.mixer.music.play()来播放，这里并没有一个类似 Music 这样的类和对象，因为背景音乐一般般只要有一个在播放就好了不是么~不放的时候就用 stop()方法来停止就好了，当然很自然有类似录影机上的 pause()和 unpause()方法。

音效和音乐方法总结

Sound 对象：

方法名	左右
fadeout	淡出声音，可接受一个数字（毫秒）作为淡出时间
get_length	获得声音文件长度，以秒计
get_num_channels	声音要播放多少次
get_volume	获取音量（0.0 ~ 1.0）
play	开始播放，返回一个 Channel 对象，失败则返回 None
set_volume	设置音量
stop	立刻停止播放

Channels 对象：

方法名	左右
fadeout	类似
get_busy	如果正在播放，返回 true
get_endevent	获取播放完毕时要做的 event，没有则为 None
get_queue	获取队列中的声音，没有则为 None
get_volume	类似
pause	暂停播放
play	类似
queue	将一个 Sound 对象加入队列，在当前声音播放完毕后播放
set_endevent	设置播放完毕时要做的 event
set_volume	类似
stop	立刻停止播放
unpause	继续播放

Sound 对象：

方法名	左右
fadeout	类似
get_endevent	类似
get_volume	类似
load	加载一个音乐文件
pause	类似

play	类似
rewind	从头开始重新播放
set_endevent	类似
set_volume	类似
stop	立刻停止播放
unpause	继续播放
get_pos	获得当前播放的位置，毫秒计

虽然很简单，不过还是提供一个例程吧，这里面音乐的播放很简单，就是上面讲过的，不过其中还有一点其他的东西，希望大家学习一下 pygame 中按钮的实现方法。

界面如上，运行的时候，脚本读取./MUSIC 下所有的 OGG 和 MP3文件（如果你不是 Windows，可能要去掉 MP3的判断），显示的也很简单，几个控制按钮，下面显示当前歌名（显示中文总是不那么方便的，如果你运行失败，请具体参考代码内的注释自己修改）：

```
# -*- coding: utf-8 -*-
# 注意文件编码也必须是 utf-8
SCREEN_SIZE=(800,600)
# 存放音乐文件的位置
```

```
MUSIC_PATH="./MUSIC"
importpygame
frompygame.localsimport*
frommathimportsqrt
importos
importos.path
defget_music(path):
# 从文件夹来读取所有的音乐文件
raw_filenames=os.listdir(path)
music_files=[]
forfilenameinraw_filenames:
# 不是 Windows 的话，还是去掉 mp3吧
iffilename.lower().endswith('.ogg')orfilename.lower().endswith('.mp3'):
music_files.append(os.path.join(MUSIC_PATH, filename))
returnsorted(music_files)
classButton(object):
"""这个类是一个按钮，具有自我渲染和判断是否被按上的功能"""
def__init__(self, image_filename, position):
self.position=position
self.image=pygame.image.load(image_filename)
defrender(self, surface):
# 家常便饭的代码了
x, y=self.position
w, h=self.image.get_size()
x-=w/2
y-=h/2
surface.blit(self.image, (x, y))
defis_over(self, point):
# 如果 point 在自身范围内，返回 True
point_x, point_y=point
x, y=self.position
w, h=self.image.get_size()
x-=w/2
y-=h/2
in_x=point_x >=xandpoint_x < x+w
in_y=point_y >=yandpoint_y < y+h
returnin_xandin_y
defrun():
pygame.mixer.pre_init(44100,16,2,1024*4)
pygame.init()
screen=pygame.display.set_mode(SCREEN_SIZE,0)
#font = pygame.font.SysFont("default_font", 50, False)
# 为了显示中文，我这里使用了这个字体，具体自己机器上的中文字体请自己查询
# 详见本系列第四部分：http://eyehere.net/2011/python-pygame-novice-professional-4/
```

```
font=pygame.font.SysFont("simsunnsimsun",50,False)
x=100
y=240
button_width=150
buttons={}
buttons["prev"]=Button("prev.png", (x, y))
buttons["pause"]=Button("pause.png", (x+button_width*1, y))
buttons["stop"]=Button("stop.png", (x+button_width*2, y))
buttons["play"]=Button("play.png", (x+button_width*3, y))
buttons["next"]=Button("next.png", (x+button_width*4, y))
music_filenames=get_music(MUSIC_PATH)
iflen(music_filenames)==0:
print"No music files found in ", MUSIC_PATH
return
white=(255,255,255)
label_surfaces=[]
# 一系列的文件名 render
forfilenameinmusic_filenames:
txt=os.path.split(filename)[-1]
print"Track:", txt
# 这是简体中文 Windows 下的文件编码，根据自己系统情况请酌情更改
txt=txt.split('.')[0].decode('gb2312')
surface=font.render(txt,True, (100,0,100))
label_surfaces.append(surface)
current_track=0
max_tracks=len(music_filenames)
pygame.mixer.music.load( music_filenames[current_track] )
clock=pygame.time.Clock()
playing=False
paused=False
# USEREVENT 是什么？请参考本系列第二部分：
# http://eyehere.net/2011/python-pygame-novice-professional-2/
TRACK_END=USEREVENT+1
pygame.mixer.music.set_endevent(TRACK_END)
whileTrue:
button_pressed=None
foreventinpygame.event.get():
ifevent.type==QUIT:
return
ifevent.type==MOUSEBUTTONDOWN:
# 判断哪个按钮被按下
forbutton_name, buttoninbuttons.iteritems():
ifbutton.is_over(event.pos):
printbutton_name,"pressed"
```

```
button_pressed=button_name
break
ifevent.type==TRACK_END:
# 如果一曲播放结束，就“模拟”按下"next"
button_pressed="next"
ifbutton_pressedisnotNone:
ifbutton_pressed=="next":
current_track=(current_track+1)%max_tracks
pygame.mixer.music.load( music_filenames[current_track] )
ifplaying:
pygame.mixer.music.play()
elifbutton_pressed=="prev":
# prev 的处理方法：
# 已经播放超过3秒，从头开始，否则就播放上一曲
ifpygame.mixer.music.get_pos() >3000:
pygame.mixer.music.stop()
pygame.mixer.music.play()
else:
current_track=(current_track-1)%max_tracks
pygame.mixer.music.load( music_filenames[current_track] )
ifplaying:
pygame.mixer.music.play()
elifbutton_pressed=="pause":
ifpaused:
pygame.mixer.music.unpause()
paused=False
else:
pygame.mixer.music.pause()
paused=True
elifbutton_pressed=="stop":
pygame.mixer.music.stop()
playing=False
elifbutton_pressed=="play":
ifpaused:
pygame.mixer.music.unpause()
paused=False
else:
ifnotplaying:
pygame.mixer.music.play()
playing=True
screen.fill(white)
# 写一下当前歌名
label=label_surfaces[current_track]
w, h=label.get_size()
```

```
screen_w=SCREEN_SIZE[0]
screen.blit(label, ((screen_w-w)/2,450))
# 画所有按钮
forbuttoninbuttons.values():
button.render(screen)
# 因为基本是不动的，这里帧率设的很低
clock.tick(5)
pygame.display.update()
if__name__=="__main__":
run()
```

```
181
182
183
184
185
```

这个程序虽然可以运行，还是很简陋，有兴趣的可以改改，比如显示播放时间/总长度，甚至更厉害一点，鼠标移动到按钮上班，按钮会产生一点变化等等，我们现在已经什么都学过了，唯一欠缺的就是实践而已！

所以下一次，我将开始一个实战篇，用 pygame 书写一个真正可以玩的游戏，敬请期待~~